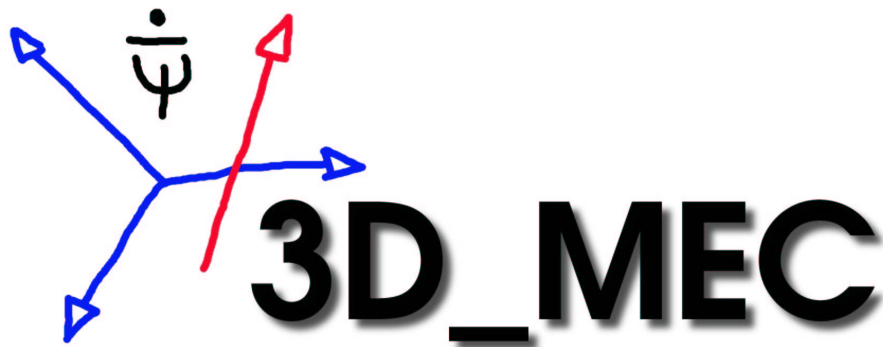


Manual de Usuario de 3D_MEC

1.99.19beta:



Programa para el manejo Numérico,
Simbólico y Gráfico de Expresiones
Vectoriales en Mecánica

Javier Ros
Dpto. Ingeniería Mecánica, Energética y de Materiales
Universidad Pública de Navarra

Noviembre 2001

Índice General

1	Introducción	1
2	El Interface de 3D_MEC. Visión general de la aplicación.	3
3	El Núcleo: Estados, Choerencia, Instrucciones.	7
3.1	Estados del Núcleo de 3D_Mec.	8
3.2	Coherencia del Núcleo de 3D_MEC.	9
3.3	Instrucciones.	12
3.3.1	Definición de Variables.	12
3.3.2	Expresiones.	16
3.3.3	Resolución de sistemas de ecuaciones no lineales.	25
3.3.4	Resolución de sistemas de ecuaciones lineales.	27
3.3.5	La instrucción PRINT	30
3.3.6	Definición de Puntos, Bases y Sistemas de Referencia.	31
3.3.7	Dibujo de diferentes elementos.	34
3.3.8	Cambio del punto de vista y del tipo de proyección.	42
3.3.9	Integración del Sistema de D.A.E.s	43
3.3.10	El Fichero de Estado.	46
3.3.11	Las Instrucciones STOP y PAUSE	50
3.4	Exportar	51

3.4.1	Exportar a MatLab	51
4	Configuración. Internacionalización.	53
4.1	Configuración	53
4.2	Internacionalización	53
5	Instalación	57
6	Disponibilidad de 3D_MEC en Internet	59
7	Limitaciones y evolución futura de 3D_MEC	61
8	Comunicación de errores	63
	Glosario	64
	Bibliografía	69

Nota de Copyright

A partir de esta versión, el programa 3D_Mec se ofrece bajo la licencia GPL de GNU. No obstante, si en el futuro parte del código de 3D_mec es transformado de forma que pueda utilizarse en forma de librería por terceros programas la licencia que deberá aplicarse a dichas librerías es LGPL de GNU.

Tal y como prescriben las citadas licencias el código debe ser distribuido junto con la aplicación o bien fácilmente accesible. Si no ha recibido el código fuente y lo desea, puede contactar al [autor](#), o bien descargarlo de la [página web](#) de la aplicación.

Por comodidad la versión actual de la licencia GPL se incluye a continuación (en inglés debido a que no existen todavía traducciones oficiales). La licencia LGPL puede encontrarse en las páginas web de la [Free Software Foundation](#).

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Soft-

ware Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright

notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the

source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this

License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Capítulo 1

Introducción

3D_Mec es una aplicación que permite realizar cálculos numéricos y simbólicos con escalares, 3-tuplas y matrices 3x3. El resultado de los cálculos puede expresarse de forma numérica, simbólica y gráfica.

Puede decirse que 3D_Mec facilita el planteamiento y análisis de problemas de simulación temporal de sistemas, o problemas de valor inicial en varias variables.

El programa ha sido diseñado para facilitar la resolución de problemas de tipo *Cinemático* y *Dinámico*, aunque también puede ser utilizado en otras disciplinas.

Es particularmente útil cuando se intentan resolver problemas de *Mecánica* en dos o tres dimensiones, utilizando las componentes de los vectores y tensores involucrados en bases de proyección arbitrarias (ver referencia [1]).

Entre las posibilidades del programa están:

- Manejo de expresiones de diferentes tipos (escalar, vectorial, 3-tupla,...) de forma numérica y simbólica.
- Definición de variables de diversos tipos y de sus derivadas temporales.
- Definición de *Puntos*, *Bases de Proyección* y de *Referencias de Observación*. Dichos elementos pueden ser referenciados en las expresiones por medio de diferentes funciones y operadores.
- Derivación temporal y paramétrica de expresiones simbólicas de todo tipo (derivación vectorial,...).
- Resolución de sistemas de ecuaciones lineales y no lineales.
- Integración numérica de sistemas de ecuaciones diferenciales (simulación).

- Representación paramétrica/variacional de un esquema del mecanismo o sistema. mediante la representación de puntos, referencias, vectores y otras entidades geométricas.
- Completa interrelación de los sistemas numérico, simbólico y gráfico. El programa mantiene la **coherencia** (sección 3.2) del **sistema** automáticamente. Se satisfacen las diferentes definiciones de variables y sistemas ecuaciones aplicados en todo momento.

Se ha puesto especial énfasis en que la entrada de datos sea natural, es decir, que sea similar al conjunto de sentencias matemáticas que normalmente se plantean en el papel a para resolver un problema.

Las operaciones matemáticas son íntegramente realizadas por el programa avisando de diferentes condiciones de error, minimizando así la posibilidad de error. La visualización gráfica de los diferentes vectores y elementos en instantes concretos o a lo largo de la evolución, ayudan a verificar la validez del planteamiento de una forma ágil.

La simulación junto con las capacidades gráficas, permite extender la práctica de la *Mecánica* hasta niveles difíciles de alcanzar con el tradicional uso del lápiz y el papel.

El conjunto de primitivas disponible permite la simulación de sistemas mecánicos de caracter muy general y la utilización de diversos formalismos.

De esta forma se consigue que el usuario pueda utilizar 3D_Mec de una forma ágil y con un mínimo de aprendizaje, facilitando la resolución de problemas mecánicos.

Capítulo 2

El Interface de 3D_MEC. Visión general de la aplicación.

En este capítulo se describe brevemente el *interface* de 3D_Mec, que es la parte del programa con la que entra en contacto el usuario y le permite interactuar con el **Núcleo de 3D_MEC** (sección 3) .

La descripción de los diferentes Menús, características del Interface así como su posible apariencia, no es exhaustiva. Tan solo se pretende dar una idea de sus posibilidades. Esto no supone ningún problema ya que el *interface* de 3D_Mec trata de ser autoexplicativo.

Los nombres de las diferentes entradas de los menús pueden cambiar entre versiones, por ello esta es una descripción conceptual. Ya que las diferentes posibilidades descritas estarán, de una forma u otra, disponibles en el interface.

Visión general:

Al ejecutar 3D_Mec lo primero que se encuentra es la VENTANA PRINCIPAL de la aplicación vacía (Figura 2.1).

Para comenzar a trabajar es necesario abrir una ventana en la que se introducen los datos. Para ello se selecciona en el Menú **Archivo** la entrada **Nuevo**, o bien la entrada **Abrir** (si se desea trabajar con un archivo existente). Aparecerá una nueva ventana (Figura 2.2):

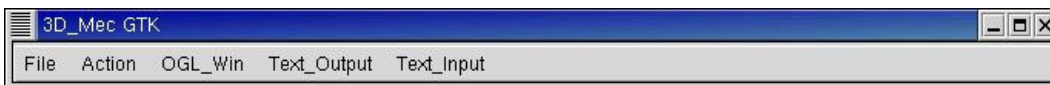
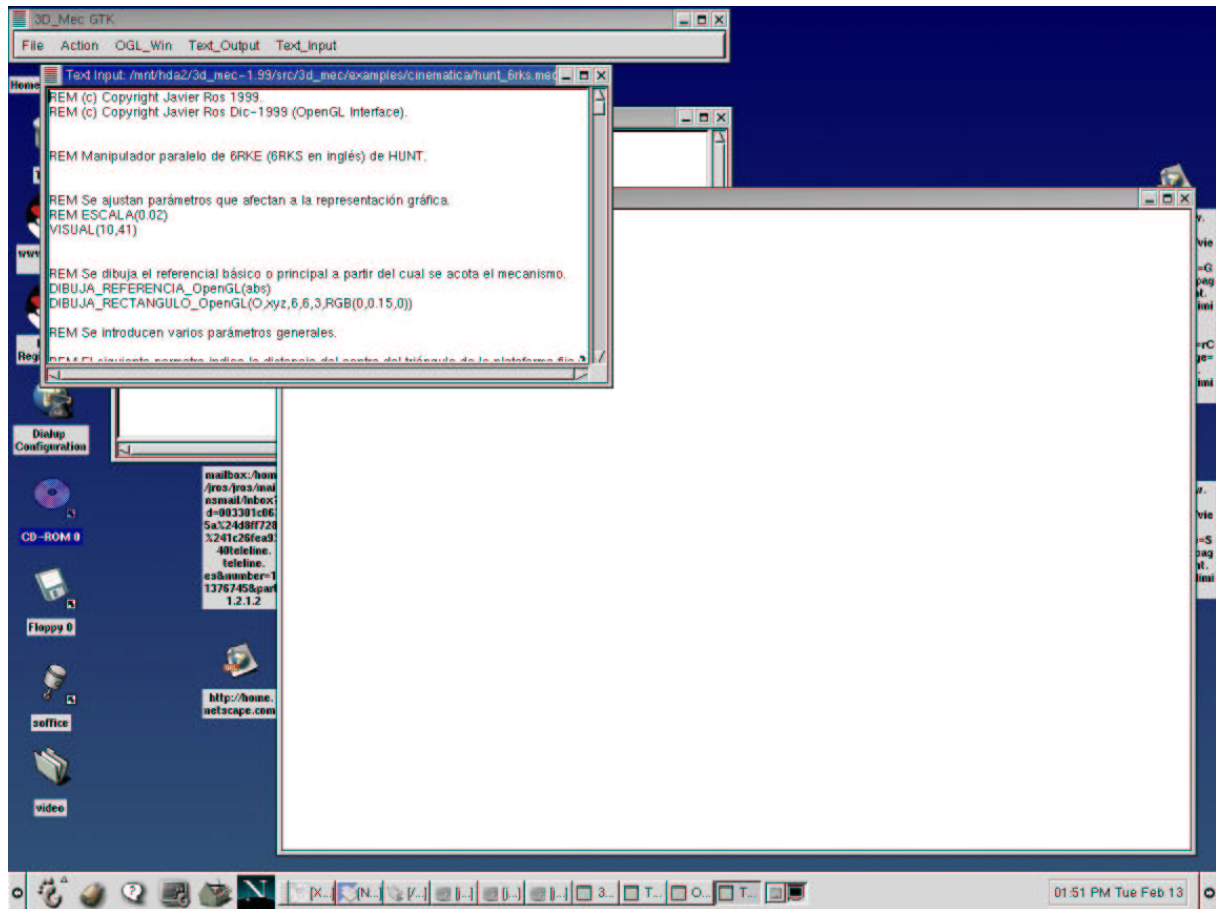


Figura 2.1: Apariencia inicial de 3D_Mec.

Figura 2.2: 3D_Mec después de abrir el *Fichero de Entrada*.

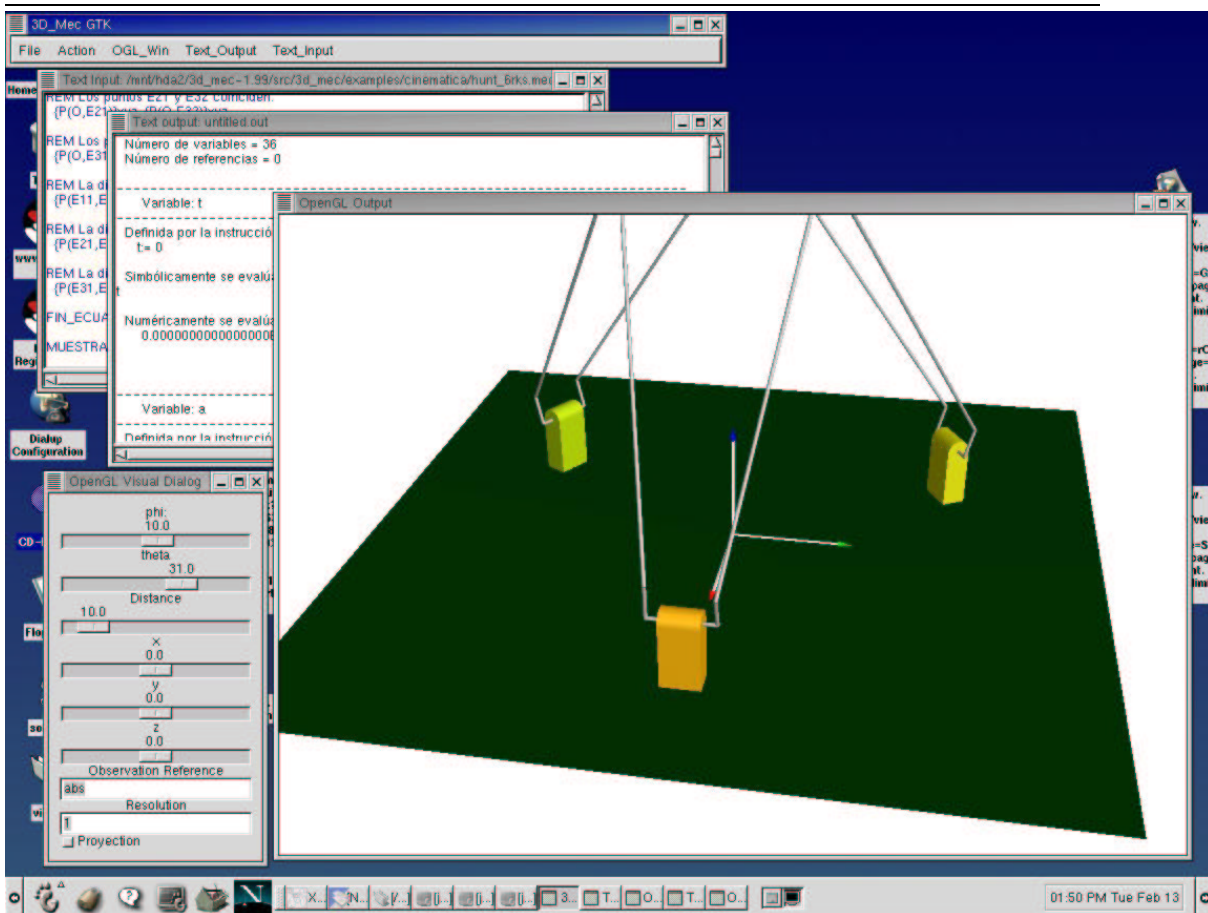


Figura 2.3: 3D_Mec después de ejecutar parte del *Fichero de Entrada*.

- VENTANA DE ENTRADA: Es un editor de texto que permite editar el denominado *Fichero de Entrada*. Este fichero contiene las **instrucciones** (sección 3.3) que ejecuta el *núcleo* para resolver un problema concreto.

Para que el *núcleo* ejecute las instrucciones introducidas en el *Fichero de Entrada*, se selecciona en el Menú **Ejecucion** la entrada **Play**. Aparecerán dos nuevas ventanas (Figura 2.3):

- VENTANA DE SALIDA: Es un editor de texto que presenta los resultados numéricos y analíticos que el *núcleo* escribe en el *Fichero de Salida*. Según la versión del *interface* de 3D_Mec los mensajes de error o de aviso que pueden producirse al ejecutar las **Instrucciones** (sección 3.3) pueden ser dirigidos también a esta ventana.
- VENTANA SALIDA GRÁFICA: Es una ventana que presenta la salida gráfica que el *núcleo* escribe en el *Dispositivo Gráfico*. La salida gráfica está

constituida generalmente por un esquema tridimensional del *Mecanismo o Sistema* con el que se está trabajando, en cuyo contexto se dibujan los vectores que se han introducido o que han aparecido como resultado del proceso de cálculo.

Capítulo 3

El Núcleo: Estados, Choerencia, Instrucciones.

El *núcleo* es la parte de la aplicación que realiza los cálculos. El Usuario utiliza el **Interface de 3D_MEC** (sección 2) para interaccionar con éste.

En esta sección se describirán únicamente aquellos aspectos del *núcleo* que son necesarios, desde el punto de vista del Usuario, para trabajar con 3D_Mec.

De una forma muy simplista puede decirse que el *núcleo* ejecuta el conjunto de **Instrucciones** (sección 3.3) contenidas en el *Fichero de Entrada*, y produce:

- Resultados que son dirigidos al *Fichero de Salida*, si son de tipo texto.
- Resultados que son dirigidos al *Dispositivo Gráfico* (Pantalla o Ventana, impresora...) si son de tipo gráfico.
- Errores o Avisos que son dirigidos al *Fichero de Salida de Errores*.

El *Fichero de Entrada*, el *Fichero de Salida*, el *Fichero de Salida de Errores* y el *Dispositivo Gráfico* tienen asociado su correspondiente elemento en el **Inteface de 3D_MEC** (sección 2) que permite su manipulación, edición, visualización, lectura, etc.. Deliberadamente se hace referencia a estos Ficheros y Dispositivos de una forma abstracta, ya que es así como son vistos o entendidos por el *núcleo* y no dependen de una versión o implementación particular del *interface* de 3D_Mec.

3.1 Estados del Núcleo de 3D_Mec.

El *núcleo* puede encontrarse principalmente en tres estados:

- **STOP:** Puede decirse que en este estado el *núcleo* se encuentra recién Inicializado, o Reinicializado. Si el *núcleo* está en este estado la única acción que puede tomarse es iniciar la ejecución de Instrucciones del *Fichero de estado* a partir de la primera de ellas.

Este es el estado en que se encuentra el *núcleo* al arrancar la aplicación. También se puede llegar a este estado mientras se están ejecutando las Instrucciones:

- por la ejecución de una **Instrucción STOP** (sección 3.3.11)
- activando en el *interface* de 3D_Mec la entrada de Menú **Ejecutar** → **Stop**.

- **PAUSE:** En este estado la ejecución de instrucciones se encuentra detenida. Es posible Continuar la ejecución de Instrucciones a partir del punto en que quedó detenida. Para ello se activa en el *interface* de 3D_Mec la entrada de Menú **Ejecutar** | **Continuar**.

Este es el estado en que se sitúa el *núcleo* cuando:

- ocurre un error en la ejecución de una Instrucción.
- se ejecuta una **Instrucción PAUSE** (sección 3.3.11) .
- se activa en el *interface* de 3D_Mec la entrada de Menú **Ejecutar** → **Pausa**.

- **PROCESANDO:** Este es el estado en que se encuentra el *núcleo* cuando está ejecutando secuencialmente las Instrucciones del *Fichero de Entrada*. Como se ha comentado más arriba, es posible detener la ejecución de Instrucciones. Las formas de Detener la ejecución pueden diferenciarse en: las que dejan al *núcleo* en el estado **PAUSE** y las que dejan a éste en el estado **STOP**.

A este estado se accede:

- desde el *interface* de 3D_Mec activando la entrada de Menú **Ejecutar** → **Play**, si el *núcleo* está en el estado **STOP**.
- desde el *interface* de 3D_Mec activando la entrada de Menú **Ejecutar** → **Pausa**, si el *núcleo* está en el estado **PAUSE**.

3.2 Coherencia del Núcleo de 3D_MEC.

El *núcleo* de 3D_Mec es capaz de evaluar **expresiones** (sección 3.3.2) matemáticas de diferente tipo. Estos tipos pueden clasificarse en: *escalar*, *3-tupla*, *vector*, *matriz 3x3* y *tensor*.

3D_Mec permite el planteamiento de un problema, o definición de un sistema mediante la **definición de variables** (sección 3.3.1) a través de expresiones matemáticas. El *núcleo* hace que las variables tomen en todo momento valores de forma que se ajusten con las expresiones simbólicas que las definen, a no ser que sean de tipo elemental (las variables elementales se definen posteriormente). Así las expresiones que definen una variable pueden considerarse como la ecuación que las define y el núcleo hace que dicha ecuación se satisfaga en todo momento (coherencia de las variables con las ecuaciones que las definen).

También es posible forzar que un conjunto determinado de variables elementales satisfagan un determinado **sistema de ecuaciones lineal** (sección 3.3.4) o **no lineal** (sección 3.3.3), el *núcleo* se encarga automáticamente de hacer que se satisfagan dichas relaciones en todo momento (coherencia de las variables con los sistemas de ecuaciones aplicados).

Al margen de variables es posible **definir** (sección 3.3.6) los siguientes objetos geométricos: *puntos*, *bases* y *referencias*. Sus definiciones pueden ser completamente parametrizadas en función de las variables definidas. Apoyándose en los elementos anteriores es posible **dibujar** (sección 3.3.7) diferentes elementos gráficos. El *núcleo* se encarga automáticamente de hacer que los objetos geométricos y los diferentes elementos gráficos satisfagan las definiciones parametrizadas que los definen (coherencia de los elementos con las expresiones que los definen).

Para 3D_Mec, el conjunto de variables definidas por el usuario, el conjunto de ecuaciones que satisfacen y los diferentes objetos geométricos y elementos gráficos definidos constituyen conceptualmente un “Sistema”. Es posible establecer una analogía entre el concepto de sistema para 3D_Mec y el sistema real que se pretende analizar. El usuario define diferentes elementos (variables, objetos...) y fuerza a que satisfagan diferentes normas (diferentes ecuaciones). Un sistema real (por ejemplo un mecanismo), es construido mediante diferentes elementos (piezas, sólidos) que están obligados a satisfacer determinadas normas (están enlazados de alguna manera y/o son impulsados con mayor o menor intensidad, cumplen las leyes de *Newton*).

3D_Mec tiene definida internamente una variable especial, la **variable tiempo** (sección 3.3.1), que hace de reloj o cronómetro del sistema. Se puede hacer que esta variable evolucione utilizando los **procedimientos de integración** (sección 3.3.9). El núcleo de 3D_Mec mantiene la coherencia del sistema durante

la evolución de la variable, con lo que en efecto se consigue realizar una simulación del sistema.

Para profundizar en la forma en que el *núcleo* mantiene la coherencia del *sistema* es necesario introducir una nueva clasificación de las variables en dos tipos: elemental y no elemental.

Se dice que una variable escalar es *elemental* cuando la expresión que la ha definido a través de una **instrucción de asignación** (sección 3.3.1) es de tipo escalar y no contiene referencias a otras variables una vez simplificada. Dicho tipo de asignación será denominado asignación elemental. Si la asignación que define una variable es no elemental, la variable es no elemental. Las variables no escalares son siempre no elementales.

Las incógnitas de los sistemas de ecuaciones pueden ser únicamente variables de tipo elemental.

Las variables no elementales no son redefinibles. Las variables elementales son sólo redefinibles mediante expresiones de tipo escalar si no han participado como incógnitas en un sistema de ecuaciones.

Las variables elementales que hayan participado como incógnitas en un sistema de ecuaciones no pueden participar como incógnitas en un nuevo sistema de ecuaciones.

Una variable elemental tiene asociada como representación simbólica su propio nombre, aunque su valor sea calculado a través de un sistema de ecuaciones. Las variables no elementales se evalúan simbólicamente a una expresión simbólica equivalente a la expresión utilizada para definir las, pero expresada en términos de variables elementales.

Todos estos hechos son normalmente transparentes al usuario. El *núcleo* avisa de cualquier eventualidad que viole esta norma impidiendo que se procese la instrucción correspondiente.

Las condiciones anteriores sobre la definición de variables y su posterior modificabilidad no suponen una gran limitación, de hecho las complejas definiciones y condiciones comentadas anteriormente describen, desde un punto de vista formal, la forma más usual y clara de planteamiento de problemas de valor inicial en varias variables. Los sistemas mecánicos caen dentro de dicha categoría, en este contexto las variables elementales representan normalmente el *estado* del sistema, los parámetros cinemáticos y dinámicos que lo definen (en forma escalar), y las acciones de enlace (expresadas también en forma escalar).

De esta forma el *núcleo* obliga de alguna manera a que el usuario plantee or-

denadamente el problema y a expresar los resultados simbólicos en términos de variables elementales. Así, las diferentes variables mantienen su significado físico en todo el contexto del problema, y se evita que el usuario las defina de una forma diferente; lo que normalmente cambiaría el significado de la variable (práctica habitual en otros contextos programáticos).

El **Núcleo de 3D_MEC** (sección 3) mantiene en todo momento la coherencia entre los valores de todas las variables y las expresiones o ecuaciones que las relacionan, liberando al usuario de esta tediosa labor. En cualquier instante de tiempo se satisfacen las relaciones entre variable y expresión no elemental a través de las cuales han sido definidas; las variables elementales que son incógnitas de los sistemas de ecuaciones toman los valores numéricos que hacen que se satisfagan dichos sistemas; y cuando varía el tiempo a través de los procedimientos de integración temporal se modifican los valores del resto de variables elementales de forma que sean coherentes con los valores de sus variables derivadas e integrales temporales (si han sido definidas).

EJEMPLO:

```
alpha := c(25.3) + 2
D(alpha) := alpha'
phi := 0.0E+0
theta := 1.0 + 2
REM alpha, alpha', phi y theta son variables elementales.
gamma := alfa * c(phi)
REM gamma no es una variable elemental.
LINEAR_EQUATION_SYSTEM(alpha')
alpha' + 37 * c(gamma) = 0
END
REM La variable alpha' no puede ser redefinida.
EULER(0.1,0.1,0.1,0)
REM La variable tiempo evoluciona, alpha' actualiza su valor
REM coherentemente con el sistema de ecuaciones del que es incognita.
REM La variable alpha modifica su valor coherentemente con su derivada
REM utilizando el método de integración numérico de EULER.
REM La variable gamma modifica su valor coherentemente con la
REM expresión no elemental que la define.
REM Las variables phi y theta no modifican su valor
REM Ya que son elementales no son incógnitas de ningún sistema de
REM ecuaciones, y no tienen definida derivada.
```

Nota: Por el momento una variable elemental que no es incógnita de un sistema de ecuaciones modifica su valor exclusivamente si tiene asociadas variables derivada

cuyo valor numérico es diferente de 0. En el caso particular en que la variable en cuestión tiene variable integral asociada de tipo elemental y que el *núcleo* conozca como varía la variable integral asociada, porque sea incógnita de un sistema de ecuaciones, su variable derivada no es modificada coherentemente. Esta limitación será corregida en breve. No obstante, la forma natural en la que los problemas de valor inicial son planteados no hace necesaria usualmente dicha funcionalidad. Podría decirse que solo es necesaria si el usuario decide hacer cosas “raras”. Además dicho problema puede solventarse asociando una expresión para dicha variable o introduciendo un nuevo sistema de ecuaciones que permita determinarla. Para entender la situación que se describe, si en el ejemplo anterior hubiese sido definida la variable derivada de `alpha'` según la instrucción de asignación `D(alpha'):=alpha'` antes del proceso de integración, la variable `alpha'` no modificaría su valor. Esto haría que perdiera la coherencia con el valor de su integral.

3.3 Instrucciones.

3.3.1 Definición de Variables.

Se definen mediante una de las siguientes *Instrucciones de Asignación*:

```
nombre_variable := expresion
{ nombre_variable } nombre_base := expresion
```

Donde:

- **nombre_variable**: Es el nombre elegido para la variable que se quiere definir y está formado por prácticamente cualquier combinación de caracteres. Debe tenerse en cuenta que existe distinción entre minúsculas y mayúsculas.
- **:=**: Es el denominado *Operador de Asignación*.
- **expresion**: Es una fórmula o sentencia matemática formada por: números, 3-tuplas, matrices 3x3, variables de cualquier tipo, funciones y los correspondientes operadores que actúan entre o sobre ellos. (véase **Expresiones** (sección 3.3.2)). El tipo de la variable definida viene dado por el tipo de **expresion**. En el segundo tipo de asignación, cuando se utilizan las llaves junto con el nombre de la base de proyección **nombre_base**, es utilizado para definir el vector o tensor a partir de una **expresión** de tipo 3-tupla o matriz 3x3 con base asociada **nombre_base**, respectivamente.

El **Núcleo de 3D_MEC** (sección 3) se encarga de mantener continuamente la **coherencia** (sección 3.2) entre los valores de las variables y los valores de las expresiones que las definen en todo momento.

Variables Escalares.

Si la expresión, **expresion**, es de **tipo** (sección 3.3.2) *escalar* entonces la variable definida es de tipo *escalar*.

Según como se definan las variables escalares estas pueden ser de tipo **elemental** (sección 3.2) .

EJEMPLO:

```
alpha := 15
Alpha := 0.0E+0
gamma := alfa * c(Alpha)
REM alpha y phi son variables elementales.
REM gamma no es elemental y su valor numérico es 15.
```

La Variable Tiempo

Por defecto el programa tiene definida una variable con significado físico de tiempo. El *núcleo* se encarga de mantener automáticamente la **coherencia** (sección 3.2) del *sistema* de variables y gráficos cuando se hace variar el tiempo mediante los procedimientos de **integración** (sección 3.3.9))

Se puede hacer referencia a ésta variable como si fuera cualquier otra variable, incluso cambiar su valor inicial a través de una Instrucción de Asignación. No obstante, no tiene sentido definir variables derivada con respecto al tiempo de la variable tiempo, ni redefinir la variable tiempo en términos de otras variables.

El nombre de la variable tiempo es **t**, e inicialmente el valor de esta variable es 0.

Variable Derivada respecto del tiempo de una variable escalar

Es posible definir variables escalares con significado de derivada con respecto al tiempo de un variable escalar dada. El *núcleo* tendrá en cuenta la relación existente entre variables y sus derivadas y mantendrá la coherencia entre estas de una forma automática

Esto permitirá aplicar el **Operador Derivada temporal** (sección 3.3.2) , sobre expresiones en las que aparezca la variable dada y producir resultados *Numéricos* y *Simbólicos* correctos.

Esta información es utilizada por el *núcleo* y en particular por los procedimientos de **integración temporal** (sección 3.3.9) , de forma que al calcular la evolución temporal las variables mantienen la **coherencia** (sección 3.2) con sus derivadas.

Si una variable varía respecto al tiempo, solamente es necesario definir una variable derivada temporal de ésta, si la primera variable va a aparecer en alguna expresión sobre la que actúe el operador **derivada temporal** (sección 3.3.2) o la variable derivada debe aparecer explícitamente en alguna expresión.

Por ejemplo, en mecánica solo suelen derivarse con respecto al tiempo las coordenadas generalizadas y las velocidades generalizadas, mientras que las acciones de enlace o de otro tipo, aunque puedan variar con el tiempo, no son afectadas por el operador derivada temporal o bien sus derivadas con respecto al tiempo no aparecen en relaciones de tipo mecánico. Esto hace que para resolver problemas de tipo mecánico, usualmente baste con definir las derivadas primera y segunda de las coordenadas generalizadas.

La sintaxis para definir una variable de tal tipo es:

```
D(nombre_variable):=nombre_derivada
```

donde `nombre_derivada` es el nombre que se utiliza para denominar a la derivada de `nombre_variable`.

Si la variable, `nombre_variable`, es **elemental** (sección 3.2) , la variable derivada, `nombre_derivada`, se inicializa a cero y será de tipo elemental. Si la variable, `nombre_variable`, no es elemental la derivada, `nombre_derivada`, no es elemental y se inicializa con la derivada de la expresión que definía la variable, `nombre_variable`.

EJEMPLO:

```
alpha := 15
D(alpha) := alphader
REM alphader vale 0 y es elemental
alphader := c(beta) *beta'
REM alphader ya no es elemental
D(alphader) := alphaderder
REM la variable alphaderder no es elemental,
REM su valor simbólico es -s(beta)*beta'^2+c(beta)*beta■
REM suponiendo que beta' y beta■ se hayan definido como derivada
```

REM primera y segunda respecto del tiempo de beta, respectivamente.

Si una variable **elemental** (sección 3.2) , que tiene asociada una variable derivada elemental, es redefinida a través de una expresión no elemental, la variable derivada queda automáticamente redefinida mediante la derivada de dicha expresión. Esta regla se extiende a las derivadas sucesivas. Naturalmente la variable y sus derivadas sucesivas dejan de ser elementales.

Variables de tipo 3-tupla y de tipo Vector

Si se utiliza el primer tipo de asignación y la expresión **expresion** es de **tipo** (sección 3.3.2) *3-tupla* entonces la variable definida es de *tipo 3-tupla*.

EJEMPLO:

```
Tupla1 := [0,L,0]
REM Tupla1 es una variable de tipo 3-tupla que no tiene asociada una base.
Tupla1_en_base_123 := { [0,L,0] } 123
REM Tupla1_en_base_123 representa la misma tupla que antes pero ahora
REM tiene asociada la base 123. Naturalmente la base 123 ha tenido que ser
REM definida con anterioridad. Nótese que se ha hecho uso del
REM operador { expresion } base .
Vabs(P)_en_base_123 := { [0, 0, alfader] } 123 ^ Tupla1_en base_123
REM Define la 3-tupla Vabs(P)_en_base_123 con base asociada 123.
Aabs(P)_en_base_123 := D({Vabs(P)}123) + {0Mabs(123)}123 ^ {Vabs(P)}123
```

Generalmente resulta más cómodo definir vectores en vez de 3-tuplas. Para ello pueden utilizarse ambos tipos de asignaciones. El primero permite definir vectores en términos de vectores. El segundo permite definir vectores a partir de 3-tuplas con base asociada.

```
{ OP } 123 := { [0, L,0] } 123 + { [0, 0, R] } 123
REM OP es una variable de tipo vector, internamente estará representada
REM por una tupla en base 123, aunque esto resulta transparente al usuario.
REM Si OP es el vector de posición que va desde el punto 0 hasta el P
REM No es necesario definirlo ya que puede sustituirse por la función
REM vector de posición P( 0, P). Siempre y cuando estén definidos
REM los puntos 0 y P. Esta suele ser la forma más comoda de trabajar.
Tupla1_en_base_1'2'3' := { OP } 1'2'3'
REM Tupla1_en_base_1'2'3' representa la misma tupla que anteriormente pero ahora
REM tiene asociada la base 1'2'3'. Nótese que se ha hecho uso del operador
```

REM { expresion } base , y que { OP } 1'2'3' es por tanto una 3-tupla que
REM representa al vector OP en base 1'2'3' .

Vabs(P) := OM(abs,solido) ^ OP

REM Define el vector Vabs(P). Nótese que se ha hecho uso del operador
REM OM(orientacion_origen, orientacion_destino).

Aabs(P) := D(Vabs(P), abs)

REM Nótese que se ha hecho uso del operador D(vector, referencia) que
REM tiene como resultado la derivada temporal del vector
REM en la correspondiente referencia.

Revísese las secciones **operadores** (sección 3.3.2) y **funciones** (sección 3.3.2) .

Variables del tipo matriz 3 x 3 y de tipo Tensor

Si se utiliza el primer tipo de asignación y la expresión **expresion** es de **tipo**
(sección 3.3.2) *matriz 3x3* entonces la variable definida es de *tipo matriz 3x3*.

EJEMPLO:

matriz_IG(disco)_123 := { [[2 * I, 0, 0] [0, I, 0] [0, 0, I]] } 123

matriz_IBsteiner(disco)_1'2'3' := { [[m * L^2, 0, 0] [0, 0, 0] [0, 0, m * L^2]] } 1'2'3'

matriz_IB(disco)_123 := matriz_IG(disco)_123 + { matriz_IBsteiner(disco)_1'2'3' } 123

Generalmente resulta más cómodo definir tensores en vez de matrices 3x3. Para
ello pueden utilizarse ambos tipos de asignaciones. El primero permite definir tensores
en términos expresiones de tipo tensor. El segundo permite definir tensores
a partir de matrices 3x3 con base asociada.

EJEMPLO:

{IG(disco)}123 := { [[2 * I, 0, 0] [0, I, 0] [0, 0, I]] } 123

{IBsteiner(disco)}1'2'3' := { [[m * L^2, 0, 0] [0, 0, 0] [0, 0, m * L^2]] } 1'2'3'

IB(disco) := IG(disco) + IBsteiner(disco)

Revísese las secciones **operadores** (sección 3.3.2) y **funciones** (sección 3.3.2) .

3.3.2 Expresiones.

En el contexto de 3D_Mec una expresión es una sentencia matemática evaluable
por el **Núcleo de 3D_MEC** (sección 3) y cuyo resultado es un número, una
3-tupla, un **vector**, una **matriz-3x3** o un **tensor**.

Cuando el *núcleo* evalúa una expresión se calculan dos resultados:

- Un resultado que se denominará *Numérico* y que estará constituido por el valor numérico resultado de la evaluación.
- Un resultado que se denominará *Simbólico* y que será el resultado de transformar la expresión en otra expresión *Simbólica* equivalente, expresada en términos de *Variables elementales* (sección 3.2) .

La instrucción **PRINT** (sección 3.3.5) puede ser utilizada para visualizar el resultado de evaluar una expresión.

NOTA: El *núcleo* tan solo es capaz de operar con *Números Reales*.

Tipos de Expresiones.

Expresiones de tipo *Escalar*:

En el contexto de 3D_Mec se dice que una **Expresión** es de tipo *Escalar* si el resultado de su evaluación es un número.

Para centrar ideas se presentan varios ejemplos de expresiones de tipo *Escalar*:

EJEMPLOS:

$1.0 + 2.0$

$1 + 2.5 * A$

$1.5 / (A^{2.0E+1} - b)$

En los ejemplos anteriores:

- La primera expresión calcula la suma de los números 1.0 y 2.0, su resultado es un número y por tanto es de tipo *Escalar*.
- La segunda suma el número 1 al resultado de multiplicar el número 2.5 por la variable A: Evidentemente la variable A ha debido ser definida con anterioridad al cálculo de la expresión, y además la variable A debe ser de tipo *Escalar*.

- La tercera divide el numero 1.5 por el resultado de elevar la variable **A** al exponente 2.0E+1 y restarle la variable **b**. Evidentemente las variables **A** y **b** deben haber sido definidas con anterioridad al cálculo de la expresión, además las variables **A** y **b** deben ser de tipo *Escalar*.

Nota: Las secciones **Operadores** (sección 3.3.2) y **Funciones** (sección 3.3.2) describen los diferentes tipos de *Operadores* y funciones definidos en 3D_Mec.

Expresiones de tipo *3-tupla*:

En el contexto de 3D_Mec se dice que una expresión es de tipo *3-tupla* si el resultado de su evaluación es una *3-tupla*.

3D_Mec interpreta todas las *3-tupla* como *3-tupla columna*.

Para centrar ideas se presentan varios ejemplos de expresiones de tipo *3-tupla*:

EJEMPLOS:

```
[1.0, -3, 2]
2.5 * A + [-5 * a, -3.02E-32, 2+2^2.0E+1]
A / (2 + b * A)
{A}123 + B
```

En los ejemplos anteriores:

- La primera expresión es un ejemplo de la expresión de tipo *3-tupla* más sencillo que puede darse. Se utilizan los corchetes para abrazar las componentes de la *3-tupla*, que se separan mediante comas. En este caso la expresión no tiene asociada una base.
- La segunda será gramaticalmente correcta sólo si las variables **A** y **a** se definen con anterioridad al cálculo de la expresión y además **A** y **a** son de tipo *3-tupla* sin base asociada y escalar, respectivamente. El resultado es la suma del producto del numero 2.5 por la variable **A** con la *3-tupla* $[-5 * a, -3.02E-32, 2+2^{2.0E+1}]$. La expresión no tiene base asociada.
- La tercera será de tipo *3-tupla* solo si la variable **A** es de tipo *3-tupla*. Además para que sea gramaticalmente correcta la variable **b** debe ser también de tipo *3-tupla*. Si las variables **A** y **b** no tienen base asociada la expresión tampoco la tendrá. Si tienen alguna base asociada, esta debe ser la misma y el resultado quedará expresado en dicha base. Evidentemente las variables **A** y **b** deben haber sido definidas con anterioridad al cálculo de la expresión.

- En el cuarto caso, si A y B son 3-tuplas y B tiene asociada la base 123, la expresión resultante es de tipo 3-tupla con base asociada 123. A puede tener asociada cualquier base o ninguna, ya que se hace uso de **operador** (sección 3.3.2) llaves.

Nota: Las secciones **Operadores** (sección 3.3.2) y **Funciones** (sección 3.3.2) describen los diferentes tipos de *Operadores* y funciones definidos en 3D_Mec.

Expresiones de tipo *vector*:

En el contexto de 3D_Mec se dice que una expresión es de tipo *vector* si el resultado de su evaluación es un *vector*. Debe notarse que en **3D_Mec** los vectores son siempre de dimensión 3, véase (sección 3.3.1).

Para centrar ideas se presentan varios ejemplos de expresiones de tipo *vector*:

EJEMPLOS:

$A \wedge B$
 $I * B$
 $P(A, B) + A$

En los ejemplos anteriores:

- La primera expresión es de tipo *vector* si A y B son de tipo *vector*.
- La segunda expresión es de tipo *vector* si I es de tipo *tensor* y B es de tipo *vector*.
- la tercera expresión es de tipo *vector* si en $P(A, B)$ A y B son puntos y en la segunda aparición de A esta es una variable de tipo *vector*.

Nota: Las secciones **Operadores** (sección 3.3.2) y **Funciones** (sección 3.3.2) describen los diferentes tipos de *Operadores* y funciones definidos en 3D_Mec.

Expresiones de tipo *matriz-3x3*.

En el contexto de 3D_Mec se dice que una expresión es de tipo *Matriz-3x3* si el resultado de su evaluación es una *Matriz-3x3*.

3D_Mec interpreta todas las *3-tupla* como *3-tupla columna*, una *matriz-3x3* solo permite el producto con una *3-tupla* por la derecha (ver sección **Operadores** (sección 3.3.2)).

Para centrar ideas se presentan varios ejemplos de expresiones de tipo *Matriz-3x3*:

EJEMPLOS:

```
[[1.0, -3, 2][1, 0, 0.0][1, -2, -7.2]]
2E+03 * A * B + [[1, 0, 0][0, 1, 0][0, 0, 1]]
```

En los ejemplos anteriores:

- La primera es un ejemplo de expresión tipo *Matriz-3x3* de lo más sencillo que puede darse. Se utilizan los corchetes para abrazar las tres *3-tupla* que representan respectivamente la primera, segunda y tercera **filas** de la *Matriz-3x3*.
- En la segunda, si A y B son *Variables* de tipo *Matriz-3x3* que han sido definidas previamente entonces, la expresión es de tipo *Matriz-3x3*.

Nota: Las secciones **Operadores** (sección 3.3.2) y **Funciones** (sección 3.3.2) describen los diferentes tipos de *Operadores* y funciones definidos en 3D_Mec.

Expresiones de tipo Tensor.

En el contexto de 3D_Mec se dice que una expresión es de tipo *tensor* si el resultado de su de su evaluación es de tipo tensor. Debe notarse que en **3D_Mec** los tensores son siempre de dimensión 3x3, véase (sección 3.3.1).

Para centrar ideas se presentan varios ejemplos de expresiones de tipo *Tensor*:

EJEMPLOS:

```
A+B
A*B
```

En los ejemplos anteriores:

- En ambos ejemplos las expresiones son de tipo *tensor* si las variables A y B son de *tensor*.

Nota: Las secciones **Operadores** (sección 3.3.2) y **Funciones** (sección 3.3.2) describen los diferentes tipos de *Operadores* y funciones definidos en 3D_Mec.

Operadores

Nombre: Derivada con respecto al tiempo.

Sintaxis: `D(expression)`

Acción: El *Operador* toma como parámetro la **Expresión** `expression` devolviendo como resultado la derivada con respecto al tiempo de ésta. En el caso de 3-tuplas y matrices esta derivada se interpreta como derivada de las componentes.

Comentarios: Puede actuar sobre cualquier **tipo de Expresión** (sección 3.3.2) exceptuando los vectores y los tensores.

Nombre: Derivada parcial.

Sintaxis: `D(expression, variable)`

Acción: El *Operador* toma como parámetro la **Expresión** `expression` y la **Variable** `variable` de *tipo escalar*, devolviendo como resultado la derivada parcial de la **Expresión** respecto de la **Variable**.

Comentarios: Puede actuar sobre cualquier **tipo de Expresión** (sección 3.3.2) exceptuando los vectores y los tensores, aunque lo más usual es utilizarla sobre expresiones de tipo escalar.

Nombre: Derivada de un vector con respecto al tiempo y respecto a una [orientación](#) ([referencia](#) o [base](#)).

Sintaxis: `D(expression, orientacion)` ó

Acción: El *Operador* toma como parámetro la **Expresión** de tipo vector `expression` y la referencia `orientacion` respecto a la que se deriva temporalmente. Devuelve como resultado la derivada con respecto al tiempo y con respecto a la referencia de dicho vector.

Comentarios: Sólo puede actuar sobre expresiones de **tipo** (sección 3.3.2) vector. En breve, la regla que implementa este operador, “Regla de derivación de vectores en referencias móviles” será extendida de forma que pueda actuar sobre 3-tuplas con base.

Nombre: Suma.

Sintaxis: `elemento1 + elemento2`

Acción: El *Operador* toma como parámetros `elemento1` y `elemento2` devolviendo como resultado su suma.

Comentarios: `elemento1` y `elemento2` deben ser del mismo tipo. Si son 3-tuplas o matrices-3x3 deben estar expresadas en la misma base y el resultado se expresa en esta.

Nombre: Diferencia.

Sintaxis: `elemento1 - elemento2`

Acción: El *Operador* toma como parámetros `elemento1` y `elemento2` devolviendo como resultado su diferencia.

Comentarios: `elemento1` y `elemento2` deben ser del mismo tipo. Si son 3-tuplas o matrices-3x3 deben estar expresadas en la misma base.

Nombre: Producto.

Sintaxis: `elemento1 * elemento2`

Acción: El *Operador* toma como parámetros `elemento1` y `elemento2` devolviendo como resultado su producto.

Comentarios: Las combinaciones posibles de tipos para `elemento1` y `elemento2`, junto con el tipo de su resultado son:

- $escalar * escalar = escalar$
- $escalar * 3-tupla = 3-tupla$ (se mantiene la base asociada a la 3-tupla)
- $escalar * matriz-3x3 = matriz-3x3$ (se mantiene la base asociada a la matriz-3x3)
- $3-tupla * escalar = 3-tupla$ (se mantiene la base asociada a la 3-tupla)
- $3-tupla * 3-tupla = escalar$ (producto escalar de dos $3-tupla$, deben estar expresadas en la misma base)
- $matriz-3x3 * escalar = matriz-3x3$ (se mantiene la base asociada a la matriz-3x3)
- $matriz-3x3 * 3-tupla = 3-tupla$ (la matriz-3x3 y la 3-tupla deben estar expresadas en la misma base que coincide con la base del resultado.)
- $matriz-3x3 * matriz-3x3 = matriz-3x3$ (las matrices-3x3 deben estar expresadas en la misma base que coincide con la base del resultado.)
- $escalar * vector = vector$
- $escalar * tensor = tensor$

- $vector * escalar = vector$
- $vector * vector = escalar$ (producto escalar de dos 3-tupla, deben estar expresadas en la misma base)
- $tensor * escalar = tensor$
- $tensor * vector = vector$
- $tensor * tensor = tensor$

Nombre: Cociente.

Sintaxis: `elemento1 / elemento2`

Acción: El *Operador* toma como parámetros `elemento1` y `elemento2` devolviendo como resultado su cociente.

Comentarios: Las combinaciones posibles de tipos para `elemento1` y `elemento2` son respectivamente:

- $escalar / escalar = escalar$
- $3-tupla / escalar = 3-tupla$
- $matriz-3x3 / escalar = matriz-3x3$
- $vector / escalar = vector$
- $tensor / escalar = tensor$

Nombre: Producto Vectorial.

Sintaxis: `elemento1 ^ elemento2`

Acción: El *Operador* toma como parámetros `elemento1` y `elemento2` y devuelve como resultado su producto vectorial.

Comentarios: Las combinaciones posibles de tipos para `elemento1` y `elemento2` son respectivamente.

- $3-tupla \wedge 3-tupla = 3-tupla$ (las 3-tuplas deben estar expresadas en la misma base, que coincide con la base del resultado)
- $vector \wedge vector = vector$

Nótese la similitud de su sintaxis con la de la función **Elevado a** (sección 3.3.2)

Nombre: Llaves.

Sintaxis: { elemento } base

Acción: Si **elemento** es de tipo vector, el resultado es una 3-tupla asociada expresada en base **base** (3-tupla con base asociada **base** resultado de proyectar el vector en la base **base**), Si **elemento** es de tipo tensor el resultado es la matriz 3x3 con base asociada **base** correspondiente. Si **elemento** es de tipo 3-tupla en una base dada, el resultado es la 3-tupla con base asociada **base** correspondiente (cambia de base la 3-tupla y le asocia la base **base**), si no tiene base asociada el operador mantiene las componentes de la tupla y le asocia la base **base**. Si **elemento** es de tipo matriz 3x3 en una base dada, el resultado es la matriz 3x3 con base asociada **base** correspondiente (cambia de base la matriz 3x3 y le asocia la base **base**), si no tiene base asociada el operador mantiene los componentes de la matriz 3x3 y le asocia la base **base**.

Comentarios: Evidentemente la base debe estar definida.

Funciones

Nombre: Paréntesis.

Sintaxis: (expresion)

Acción: Evalúa la **Expresión** *expresion*. Como es sabido los paréntesis son habitualmente utilizados para cambiar la forma en que se evalúan las expresiones, ya que gracias a ellos es posible cambiar el orden de evaluación u orden de precedencia en la evaluación.

Nombre: Seno.

Sintaxis: s(expresion)

Acción: Devuelve el seno de la **Expresión** *expresion*. El resultado es una **Expresión** de *tipo escalar*.

Comentarios: La **Expresión** *expresion* debe ser de *tipo escalar*.

Nombre: Coseno.

Sintaxis: c(expresion)

Acción: Devuelve el coseno de la **Expresión** *expresion*. El resultado es una

Expresión de *tipo escalar*.

Comentarios: La **Expresión** `expresion` debe ser de *tipo escalar*.

Nombre: Potenciación.

Sintaxis: `elemento1 ^ elemento2`

Acción: El *Operador* toma como parámetros `elemento1` y `elemento2` y devuelve como resultado `elemento1` elevado a `elemento2`.

Comentarios: `elemento1` y `elemento2` deben ser de *tipo escalar*. Además la expresión `elemento2` una vez simplificado debe ser **elemental** (sección 3.2) . Nótese la similitud de su sintaxis con la del **Producto Vectorial** (sección 3.3.2) .

Nombre: Vector de Posición.

Sintaxis: `P( punto1, punto2)`

Acción: El *Operador* toma como parámetros los **puntos** (sección 3.3.6) `punto1` y `punto2` y devuelve como resultado el vector de posición que va de `punto1` a `punto2`.

Comentarios: Los puntos `punto1` y `punto2` deben haber sido definidos con anterioridad.

Nombre: Velocidad Angular.

Sintaxis: `OM( orientacion1, orientacion2)`

Acción: El *Operador* toma como parámetros las *orientaciones* `orientacion1` y `orientacion2` y devuelve como resultado el vector velocidad angular de la `orientacion2` respecto de la `orientacion1`.

Comentarios: Las orientaciones `orientacion1` y `orientacion2` han tenido que ser definidas previamente.

3.3.3 Resolución de sistemas de ecuaciones no lineales.

La Instrucción `NON_LINEAR_EQUATION_SYSTEM` se utiliza para aplicar sistemas de ecuaciones no lineales.

Sintaxis:

`NON_LINEAR_EQUATION_SYSTEM(q1, . . . , qp)`

```

ecuacion_1
...
ecuación_g
END

```

donde:

- $q_1, \dots, q_j, \dots, q_p$ son las p variables incógnita de *tipo escalar*. Por cuestiones de **coherencia** (sección 3.2) las variables incógnita deben ser elementales pues si no estarían obligadas de antemano a satisfacer cierta relación.
- `ecuacion_1, ..., ecuacion_i, ..., ecuacion_g`, son las ecuaciones a despejar.

La sintaxis para incluir las ecuaciones es idéntica a la descrita en el apartado dedicado a los sistemas de ecuaciones lineales.

A partir de la aplicación del sistema de ecuaciones **Núcleo de 3D_MEC** (sección 3) se encarga de mantener continuamente la **coherencia** (sección 3.2) entre los valores de las variables incógnita y los sistemas de ecuaciones correspondientes.

El proceso de resolución de un sistema de ecuaciones no lineales es un proceso iterativo que, a partir de una estimación inicial de las variables incógnitas, trata de converger a una solución. Como se sabe presenta las siguientes limitaciones:

- pueden existir varias soluciones.
- la estimación inicial para las incógnitas determina la solución a la que se converge.
- normalmente se converge hacia la solución más “próxima” a la estimación inicial, aunque puede no ser así.
- si la estimación inicial no es muy buena puede ocurrir que el proceso no converja, en este caso habrá que utilizar otra estimación inicial.

El proceso iterativo de resolución se detiene por dos motivos: cuando converge o cuando el número máximo de iteraciones se ha alcanzado. La tolerancia de convergencia y el número máximo de iteraciones, están reflejados en el fichero `3d_mec.conf`.

Es posible plantear sistemas de ecuaciones no lineales compatibles e indeterminados, por ejemplo porque el número de incógnitas es superior al de ecuaciones

independientes. En estas circunstancias se resuelve el sistema para el primer y más pequeño conjunto de incógnitas para el que sea posible (el orden de precedencia es el marcado por el orden en que se especifican las incógnitas), además se producen mensajes de aviso similares a los del caso de sistemas de ecuaciones lineales *Compatibles Indeterminados*.

3.3.4 Resolución de sistemas de ecuaciones lineales.

La instrucción `LINEAR_EQUATION_SYSTEM` se utiliza para aplicar sistemas de ecuaciones lineales.

Sintaxis:

```
LINEAR_EQUATION_SYSTEM(a1,...,am)
    ecuacion_1
    ...
    ecuacion_n
END
```

donde:

- $a_1, \dots, a_j, \dots, a_m$ son las m variables incógnita de *tipo escalar*. Por cuestiones de **coherencia** (sección 3.2) , las variables incógnita deben ser elementales ya que si no estarían obligadas de antemano a satisfacer cierta relación.
- `ecuacion_1, ..., ecuacion_i, ..., ecuacion_n`, son las ecuaciones a despejar Naturalmente las ecuaciones deben ser lineales en las variables a despejar..

Por otra parte las Ecuaciones `ecuacion_i` tienen la siguiente sintaxis:

```
expresion_1 = expresion_2
```

donde ambas expresiones deben ser del mismo **tipo** (sección 3.3.2) . Además si son:

- de *tipo escalar* dan lugar a **una** única ecuación escalar.
- de *tipo 3-tupla* dan lugar a **tres** ecuaciones escalares.
- de *tipo matriz-3x3* dan lugar a **nueve** ecuaciones escalares.

- Si alguna de las ecuaciones escalares resultantes, se reduce a la ecuación escalar $0 = 0$ quedará eliminada del conjunto de ecuaciones.

A partir de la aplicación del sistema de ecuaciones **Núcleo de 3D_MEC** (sección 3) se encarga de mantener continuamente la **coherencia** (sección 3.2) entre los valores de las variables incógnita y los sistemas de ecuaciones correspondientes.

El número de ecuaciones escalares puede ser mayor, igual o menor que el numero de incógnitas, en cuyo caso cuando se analice la solución obtenida se tendrá en cuenta que:

- El sistema puede ser *Incompatible*, en cuyo caso no tiene solución y se produce un error.
- El sistema puede ser *Compatible Determinado*, lo cual implica que se ha conseguido asignar valores a todas las incógnitas propuestas utilizando el sistema de ecuaciones dado.
- Si sistema es *Compatible Indeterminado* se encuentran valores no indeterminados para un subconjunto de las incógnitas, el resto de las incógnitas queda indeterminado. Del conjunto de incógnitas indeterminado unas toman su valor en función del valor de las restantes incógnitas indeterminadas, que no modifican su valor en la solución del sistema de ecuaciones. Un mensaje de aviso comunicará cuales son las incógnitas indeterminadas y, de ellas, cuales son las incógnitas para las que no se encuentra ecuación. Evidentemente, el valor que se despeje para el conjunto de incógnitas indeterminadas para las que se encuentra ecuación depende del valor que tienen las incógnitas para las que no se encuentra ecuación. Cabe destacar que, el conjunto de variables para las que no se encuentra ecuación no es unívoco y depende del orden en que se introduzcan las incógnitas y/o las ecuaciones.

NOTA: La tercera situación es muy frecuente en la *Teoría de la Mecánica*. En esta situación se dice que algunas acciones de enlace son redundantes, o que el sistema de acciones de enlace es indeterminado. No obstante las incógnitas de movimiento (aceleraciones generalizadas) se pueden resolver siempre con éxito y por tanto pertenecerán al conjunto de incógnitas determinadas.

También este es el caso de lo problemas geométrico (aunque este es un problema en general no lineal) y cinemático, cuando el número de ecuaciones de restricción es inferior al de coordenadas/velocidades generalizadas de partida.

EJEMPLO (Mecánica): Utilización para despejar las velocidades generalizadas a partir de las ecuaciones de enlace.

Si se tienen c ecuaciones de enlace independientes o no, y se utilizan p velocidades generalizadas para plantear el problema:

```

LINEAR_EQUATION_SYSTEM (v1,...,vp)
  ecuacion_enlace_1
  ...
  ecuacion_enlace_c
END

```

En esta situación se despejan algunas velocidades generalizadas en función de otras. Como se han puesto como incógnitas todas las velocidades generalizadas, solo se despejan un número de ellas = p - número de grados de libertad. El número de grados de libertad coincide con el número de velocidades generalizadas para las que no se encuentra ecuación.

EJEMPLO (Mecánica): Utilización para despejar las aceleraciones generalizadas (ecuaciones de movimiento), y acciones de enlace (fuerzas y momentos de enlace) de las ecuaciones dinámicas.

Si se habían planteado con anterioridad las ecuaciones de enlace (se supone que se sigue con el problema anterior), en general será necesario añadir al conjunto de ecuaciones de la dinámica las derivadas de las ecuaciones de enlace. Se tendrían p aceleraciones generalizadas y c ecuaciones de enlace (independientes o no). Si además hay e acciones de enlace y d ecuaciones dinámicas (independientes o no) el planteamiento sería:

```

LINEAR_EQUATION_SYSTEM(a1,...,ap,F1,...,Fe)

```

REM a1,...,ap son las aceleraciones generalizadas asociadas a las
 REM velocidades generalizadas v1,...,vp del ejemplo anterior

```

D(ecuacion_enlace_1)
...
D(ecuacion_enlace_c)

ecuacion_dinamica_1
...
Ecuacion_dinamica_d

```

```

END

```

En esta situación el conjunto de las aceleraciones generalizadas (ecuaciones de movimiento) está determinado siempre. Si el sistema no es redundante (*Determinado*) todas las acciones de enlace están determinadas. Si es redundante

(*Compatible Indeterminado*) algunas acciones podrán estar determinadas y otras no, el *núcleo* dará un mensaje de aviso indicando el conjunto de incógnitas indeterminadas (acciones de enlace indeterminadas) y el conjunto de incógnitas para el cual no se ha encontrado ecuación (acciones de enlace cuyo valor permanecerá igual que antes de la aplicación del sistema de ecuaciones). El valor de las incógnitas indeterminadas para las que se encuentra ecuación dependerá del valor de las incógnitas para las que no se encuentra ecuación.

3.3.5 La instrucción PRINT

La instrucción PRINT permite obtener información sobre las diferentes variables, sistemas de ecuaciones planteados, etc.. Su comportamiento viene determinado por los argumentos que le acompañan. Si se introduce la instrucción PRINT sin ningún argumento, se obtiene como resultado información sobre todas las variables presentes hasta ese momento, así como información complementaria acerca del estado del núcleo: número de variables y referencias definido, etc ...

Sintaxis:

```
PRINT(expresion | LINEAR_EQUATION_SYSTEM[_num] | NON_LINEAR_EQUATION_SYSTEM[_num])
```

Si el argumento es:

- La expresión **expresion**, se muestra el resultado de su evaluación. Este resultado se muestra de forma *Numérica* y *Simbólica*.
- La palabra clave **LINEAR_EQUATION_SYSTEM**, se muestran los diferentes sistemas de ecuaciones lineales que han sido planteados hasta el momento. Las ecuaciones se muestran de forma *Simbólica* y sin despejar. Si se utiliza la palabra clave junto con su parte opcional, **LINEAR_EQUATION_SYSTEM_num** donde **num** es un número natural, solamente se presenta el sistema lineal de ecuaciones introducido en **num**-ésimo lugar.

Por ejemplo, si se quiere mostrar únicamente el primer sistema de ecuaciones lineales aplicado, la instrucción que se debe introducir es: `PRINT(LINEAR_EQUATION_SYSTEM_1)`.

Si se utiliza el modificador opcional **TRIANG** el sistema lineal se muestra triangularizado. Esto quiere decir que si se aplicasen las ecuaciones secuencialmente y en el orden en que se presentan, las variables tendrían después de terminar el valor que les correspondería si se resolviese el sistema de ecuaciones. Se puede decir que las ecuaciones se presentan despejadas para propósitos computacionales.

- La palabra clave `NON_LINEAR_EQUATION_SYSTEM`, se muestran los diferentes sistemas de ecuaciones no lineales que han sido planteados hasta el momento. Las ecuaciones se muestran de forma *Simbólica* y sin despejar. Como con la palabra clave anterior, se puede mostrar un sistema no lineal específico utilizando la parte opcional `_num`. Por ejemplo para mostrar únicamente el primer sistema de ecuaciones no lineales aplicado se escribe: `PRINT(NON_LINEAR_EQUATION_SYSTEM_1)`.

3.3.6 Definición de Puntos, Bases y Sistemas de Referencia.

Referencia Absoluta y su Base y Punto asociados.

Por defecto está definida una Referencia Absoluta. Es necesaria como referencia inicial para poder situar en el espacio los diferentes elementos. Esta referencia se denomina `abs` (en minúsculas).

También por defecto está definida la Base que tiene asociada esta referencia. Esta base determina la orientación de la Referencia Absoluta y se denomina `xyz` (en minúsculas).

Por último también está definido por defecto el Punto Origen asociado a la Referencia Absoluta. Este punto se denomina `O` (letra o en mayúsculas).

A partir de estos elementos, es posible definir nuevos puntos, bases y referencias. Además estos elementos están completamente integrados en la estructura simbólica, numérica y gráfica del *núcleo*, siendo posible hacer referencia a ellos a través de diversos operadores y funciones dentro de las expresiones. Estos elementos también son referenciados a la hora de representar gráficamente el sistema. De esta forma las instrucciones gráficas y las expresiones están interrelacionados, se minimiza la posibilidad de error por parte del usuario debido a la íntima relación que se establece entre la parte gráfica y la parte simbólico-numérica del *núcleo*.

Definición de un Punto.

Para definir un *Punto* se utiliza la instrucción `POINT`.

Para definir una nueva *Referencia* es necesario definir antes el *Punto* que posiciona su origen, también será necesario definir una base de proyección que defina su orientación.

Sintaxis:

`POINT(punto_nuevo, punto_anterior, 3-tupla_posicion)`

Donde:

- `punto_nuevo`: Nombre que se da al *Punto* que se está definiendo.
- `punto_anterior`: Nombre del *Punto* a partir del cual se define el `punto_nuevo`.
- `3-tupla_posición`: 3-tupla con base asociada que determina la posición relativa del `punto_nuevo` respecto al `punto_anterior`.

EJEMPLO:

`POINT(A, 0, { [0, 0, R] } xyz)`

Define el Punto A, a partir del punto 0 (origen de la referencia absoluta). Su posición respecto del punto O viene determinada por la 3-tupla $\{[0,0,R]\}$ xyz.

Nota: Está previsto ampliar la sintaxis de forma que la 3-tupla de posición pueda ser alternativamente de tipo vector (creo que ya funciona).

Definición de una Base.

Para definir una *Base* se utiliza la instrucción **BASE**.

Para definir una nueva *Referencia* es necesario definir antes la *Base* que determine su orientación.

Sintaxis:

`BASE(base_nueva, base_anterior, numero_eje, angulo)`

Donde:

- `base_nueva`: Nombre que se da a la *Base* que se está definiendo.
- `base_anterior`: Nombre de la *Base* a partir de la cual se define la `base_nueva`.
- `numero_eje`: Numero de eje de la `base_anterior` sobre el que se giraría ésta para obtener la `base_nueva`. Puede tomar los valores 1, 2 o 3.

- **angulo**: Ángulo con que se encuentra girada la **base_nueva** con respecto a la **base_anterior**. Este parámetro es una **Expresión** (sección 3.3.2) .

EJEMPLO:

```
BASE(base123, xyz, 3, -phi+theta)
```

Define la Base **base123**, que se encuentra girada un ángulo **-phi+theta** en torno al eje número 3 de la Base **xyz**.

Definición de una Referencia.

La instrucción REF permite definir una *Referencia*. Para definir una nueva *Referencia* es necesario definir antes la *Base* que determina su orientación y el *Punto* que posiciona su origen.

Sintaxis:

```
REF(nombre_referencia, punto_referencia, base_referencia)
```

Donde:

- **referencia_nueva**: Nombre de la *Referencia* que se define.
- **punto_referencia**: Nombre del *Punto* que determina el origen de la **referencia_nueva**.
- **base_referencia**: Nombre de la *Base* que define la orientación de la **referencia_nueva**.

EJEMPLO:

```
REF(r1, P1, abs)
```

Define la referencia **r1** con origen en el punto **P1** , cuya orientación es la de la base **abs**.

3.3.7 Dibujo de diferentes elementos.

El nombre de la instrucción básicamente determina su funcionalidad, y los parámetros que se indican en la mayor parte de los casos tienen un significado evidente. Sólomente cuando se considere oportuno se comentarán cuestiones al respecto.

La sintaxis utilizada para describir las diferentes instrucciones es:

NOMBRE_INSTRUCCION (Param_1, Param_2, ... [, Param_N])

Los corchetes agrupan parámetros o grupos de parámetros que pueden ser omitidos, y pueden ser utilizados recursivamente (p.e.: ... [,Param_j,... [,Param_k,...]])

El parámetro especial, RGB(numero_Red, numero_Green, numero_Blue [, numero_Alpha]), se utiliza para especificar el color del objeto utilizando el standard RGB. El único parámetro que necesita explicación es **numero_Alpha**, que indica la opacidad del elemento. Un valor de 1 indica que el objeto es completamente opaco: valores inferiores a 1 representan objetos con diferente grado de transparencia.

A partir de la versión 1.99.beta19, el color de un objeto puede especificarse con anterioridad de la instrucción de dibujo, mediante la utilización de la instrucción aislada RGB(numero_Red, numero_Green, numero_Blue [, numero_Alpha]), que lo que hace es cambiar el color de dibujo por defecto.

Dibujo de esferas:

DRAW_SPHERE(Punto_Centro, Base_Orientacion, Expresion_Radio
[, RGB(R,G,B)])

Dibujo de esferas:

DRAW_SPHERE_ARC(Punto, Base_Orientacion,
Expresion_Radio,
Expresion_Angulo_a_cara_Inferior, Expresion_Angulo_a_cara_Superior,
Expresion_Angulo_Inicial, Expresion_Angulo_Final,
Eje={1, 2, 3} [, Option={0, 1}] [, RGB(R,G,B)])

Instrucción análoga a DRAW_EMPTY_CYLINDER_ARC.

Los parámetros **Expresion_Angulo_Inicial** y **Expresion_Angulo_Final** indican el inicio y el final del arco de esfera en torno al eje Eje. Cuando el parámetro Eje es:

- 1 los arcos se toman a partir del eje 2
- 2 los arcos se toman a partir del eje 3
- 3 los arcos se toman a partir del eje 1

Además debe cumplirse `Expresion_Angulo_Inicial > Expresion_Angulo_Final`.

El parámetro `Expresion_Radio` indica radio de la esfera (dentro de poco se añadirá la instrucción `DRAW_EMPTY_SPHERE_ARC` que simplemente añadirá el parámetro `Expresion_Radio_Interior` para indicar el radio interior de la esfera, esta instrucción contendrá la funcionalidad del resto de instrucciones destinadas al dibujo de esferas).

Los parámetros `Expresion_Angulo_a_cara_Inferior` y `Expresion_Angulo_a_cara_Superior` indican las posiciones de las caras de corte inferior y superior de la esfera a partir del punto `Punto`, y en la dirección del eje `Eje`. Debe cumplirse la relación `Expresion_Angulo_a_cara_Inferior > Expresion_Angulo_a_cara_Superior`, aunque los valores de estos parámetros deben de estar entre ± 90 grados.

Por defecto (igual que si se especifica `opcion` igual a 1) se dibuja como sólido. Si se especifica `opcion` igual a 0 se pinta exclusivamente la superficie correspondiente a la cara exterior del arco de esfera.

Dibujo de arcos de Esfera:

```
DRAW_EMPTY_SPHERE_ARC( Punto, Base_Orientacion,  
    Expresion_Radio_Exterior, Expresion_Radio_Interior,  
    Expresion_Angulo_a_cara_Inferior, Expresion_Angulo_a_cara_Superior,  
    Expresion_Angulo_Inicial, Expresion_Angulo_Final,  
    Eje={1, 2, 3} [, Option={0, 1}] [, RGB(R,G,B)])
```

NO IMPLEMENTADA TODAVÍA: Instrucción análoga a `DRAW_EMPTY_CYLINDER_ARC` que está destinada a sustituir a `DRAW_SPHERE_ARC` y a `DRAW_SPHERE`.

Los parámetros `Expresion_Angulo_Inicial` y `Expresion_Angulo_Final` indican el inicio y el final del arco. Cuando el parámetro `eje` es:

- 1 los arcos se toman a partir del eje 2
- 2 los arcos se toman a partir del eje 3
- 3 los arcos se toman a partir del eje 1

Además debe cumplirse `Expresion_Angulo_Inicial > Expresion_Angulo_Final` (creo).

Los parámetros `Expresion_Radio_Exterior` y `Expresion_Radio_Interior` indican los radios exterior e interior de la esfera.

Los parámetros `Expresion_Angulo_a_cara_Inferior` y `Expresion_Angulo_a_cara_Superior` indican las posiciones de las caras de corte inferior y superior de la esfera a partir del punto `Punto`, y en la dirección del eje `Eje`. Debe cumplirse la relación `Expresion_Angulo_a_cara_Inferior > Expresion_Angulo_a_cara_Superior`, aunque los valores de estos parámetros deben de estar entre $+ - 90$ grados.

Por defecto (igual que si se especifica `opcion` igual a 1) se dibuja como sólido. Si se especifica `opcion` igual a 0 se pinta exclusivamente la superficie correspondiente a la cara exterior del arco de esfera.

Dibujo de elipsoides:

```
DRAW_ELLIPSOID( Punto_Centro, Base_Orientacion, Expresion_a,
                Expresion_b, Expresion_c [, RGB(R,G,B)])
```

Dibujo de Cilindros, a partir de 2 puntos extremos situados en los centros de las caras:

```
DRAW_CYLINDER_P( Punto_Inicial, Punto_Final, Expresion_Radio
                [, Option={0,1} [, RGB(R,G,B)])
```

El parametro `Option` toma el valor 0 cuando el cilindro es hueco, y 1 cuando es macizo.

Dibujo de Cilindros mazizos, a partir de 1 punto y orientación dada por una base de proyección.

```
DRAW_CYLINDER( Punto_Centro, Base_Orientacion, Expresion_Radio,
                Expresion_Longitud,Eje={1,2,3}
                [, Option={0,1} [, RGB(R,G,B)])
```

El parámetro `Eje` indica en que dirección de la base se alinea el cilindro. El parámetro opción indica si el punto `Punto_Centro` es el centro del cilindro, o bien el centro de una de las caras, la longitud `Expresion_Longitud`, puede ser negativa con lo cual el cilindro crece en la dirección negativa del eje `Eje` seleccionado.

Dibujo de Hiperboloides Cilíndricos, mazizos o huecos, a partir un punto y orientación dada por una base de proyección.

```
DRAW_CYLINDRIC_HYPERBOLOID( Punto_Centro, Base_Orientacion, Expresion_Radio,
                             Expresion_Longitud1, Expresion_Longitud2,
                             Expresion_Tangente_Angulo, Eje={1,2,3},
                             [, Option={0,1} [, RGB(R,G,B)]])
```

El parámetro **Eje** indica que dirección de la base es el eje del Hiperboloide. El parámetro opción indica si el hiperboloide es mazizo o no. Las longitudes **Expresion_Longitud1** y **Expresion_Longitud2**, pueden tomar cualquier valor siempre que **Expresion_Longitud1 < Expresion_Longitud2**. Indican las posiciones de los extremos del hiperboloide, en su eje, respecto del **Punto_Centro** que es el punto central del hiperboloide (cuello del hiperboloide).

Dibujo de troncos de Cono, a partir de 2 puntos puntos extremos situados en los centros de las caras:

```
DRAW_CONE_P( Punto_Inicial, Punto_Final, Expresion_Radio_Inicial,
              Expresion_Radio_Final [, Option={0,1} [, RGB(R,G,B)]])
```

Instrucción análoga a **DRAW_CYLINDER_P**

Dibujo de de troncos de Cono mazizos, a partir de 1 punto y orientación dada por una base de proyección.

```
DRAW_CONE( Punto_Centro, Base_Orientacion, Expresion_Radio1,
            Expresion_Radio2, Expresion_Longitud, Eje={1,2,3}
            [, Option={0,1} [, RGB(R,G,B)]])
```

Instrucción análoga a **DRAW_CYLINDER**, aunque el parámetro opción determina si es mazizo o no, ya que determinar si afecta a ser el centro o una de las caras del cono parece algo complicado. Seguramente esta instrucción cambiará en el futuro, de forma que haya 2 longitudes: anterior al centro en la dirección al eje y posterior, lo mismo parece adecuado para otros objetos, ya que confiere mayor uniformidad logica y generalidad,...

Dibujo de vectores:

```
DRAW_VECTOR( Punto_Inicio, Expresion_Vector, Expresion_Factor_Escala
              [, RGB(R,G,B)]])
```

Dibujo de Referencias:

```
DIBUJA_Referencia_OpenGL( Referencia, Expresion_Factor [, RGB(R,G,B)]])
```

Dibujo de líneas:

```
DRAW_LINE( Punto_Inicial, Punto_Final, Trama,
           Grosor [, RGB(R,G,B)])
```

El parámetro **grosor** indica el número de píxeles de anchura que tiene la línea, al quedar fijo el grosor de la línea no se escala tridimensionalmente (creo). El parámetro **trama** es un número entero que permite que la línea no sea continua sino que pueda presentar puntos con diferente espaciado (creo que va entre 0 y 255).

Dibujo de Paralelepípedos:

```
DRAW_PARALLEPIPED( Punto_Centro, Base_Orientacion, Expresion_a,
                   Expresion_b, Expresion_c [, Option=0, [, RGB(R,G,B)]] |
                   [, Option=1, Eje={1,2,3} [, RGB(R,G,B)]])
```

Instrucción análoga a `DRAW_CYLINDER`.

El parámetro **opcion** es 0 si el punto `Punto_Centro` es el centro del paralelepípedo, o bien 1 si es el centro de una de las caras. En este caso hace falta especificar también el parámetro **Eje** que indica a que pareja de caras se refiere (el eje es perpendicular a ellas). Los parámetros que dan las longitudes de los diferentes ejes del paralelepípedo pueden tomar valores negativos, esto solo es relevante para el parámetro que va en la dirección del eje seleccionado, y hace que el paralelepípedo se extienda en dirección contraria al eje considerado.

Dibujo de arcos de Cilindro:

```
DRAW_CYLINDER_ARC( Punto_Centro, Base_Orientacion, Expresion_Radio,
                   Expresion_Longitud, Expresion_Angulo_Inicial,
                   Expresion_Angulo_Final, Eje={1, 2, 3} [, RGB(R,G,B)])
```

Instrucción análoga a `DRAW_CYLINDER`.

Los parámetros `Expresion_Angulo_Inicial` y `Expresion_Angulo_Final` indican el inicio y el final del arco. Cuando el parámetro **eje** es:

- 1 los arcos se toman a partir del eje 2
- 2 los arcos se toman a partir del eje 3
- 3 los arcos se toman a partir del eje 1

Además debe cumplirse `Expresion_Angulo_Inicial > Expresion_Angulo_Final` (creo).

Dibujo de arcos de Cilindro:

```
DRAW_EMPTY_CYLINDER_ARC( Punto, Base_Orientacion,
    Expresion_Radio_Exterior, Expresion_Radio_Interior,
    Expresion_Distancia_a_cara_Inferior, Expresion_Distancia_a_cara_Superior,
    Expresion_Angulo_Inicial, Expresion_Angulo_Final,
    Eje={1, 2, 3} [, Option={0, 1}] [, RGB(R,G,B)])
```

Instrucción análoga a `DRAW_EMPTY_SPHERE_ARC` (no está implementada todavía pero incluirá las posibilidades de `DRAW_SPHERE` y `DRAW_SPHERE_ARC`, en una sola instrucción) que está destinada a sustituir a `DRAW_CYLINDER_ARC` y a `DRAW_CYLINDER`.

Los parámetros `Expresion_Angulo_Inicial` y `Expresion_Angulo_Final` indican el inicio y el final del arco. Cuando el parámetro `eje` es:

- 1 los arcos se toman a partir del eje 2
- 2 los arcos se toman a partir del eje 3
- 3 los arcos se toman a partir del eje 1

Además debe cumplirse `Expresion_Angulo_Inicial > Expresion_Angulo_Final`.

Los parámetros `Expresion_Radio_Exterior` y `Expresion_Radio_Interior` indican los radios exterior e interior del cilindro.

Los parámetros `Expresion_Distancia_a_cara_Inferior` y `Expresion_Distancia_a_cara_Superior` indican las posiciones de las caras 1 y 2 del cilindro a partir del punto `Punto`, y en la dirección del eje `Eje`. Debe cumplirse la relación `Expresion_Distancia_a_cara_Inferior > Expresion_Distancia_a_cara_Superior`, aunque los valores de estos parámetros pueden tomar cualquier signo.

Por defecto (igual que si se especifica `option` igual a 1) se dibuja como sólido. Si se especifica `opcion` igual a 0 se pinta exclusivamente la superficie correspondiente a la cara exterior del arco de cilindro.

Dibujo de Rectangulos:

```
DRAW_RECTANGLE( Punto_Centro, Base_Orientacion, Expresion_a,
    Expresion_b, Eje={1, 2, 3} [, RGB(R,G,B)])
```

El parámetro **Eje**, indica el eje perpendicular al plano del rectángulo. los parámetros **Expresion_a** y **Expresion_b** determinan los lados del rectángulo.

Cuando el parámetro **Eje** es:

- 1 el parámetro **Expresion_a** es la longitud del eje 2 y el parámetro **Expresion_b** el la longitud del eje 3
- 2 el parámetro **Expresion_a** es la longitud del eje 3 y el parámetro **Expresion_b** el la longitud del eje 1
- 3 el parámetro **Expresion_a** es la longitud del eje 1 y el parámetro **Expresion_b** el la longitud del eje 2

(creo).

Dibujo de Texto Tridimensional:

```
DRAW_TEXT( Punto_Inicial, Base_Orientacion, Eje={1, 2, 3}, Texto,
           [, Expresion_Factor_Escala [, RGB(R,G,B)]] )
```

El parámetro **Expresion_Factor_Escala** multiplica al tamaño de la fuente por defecto (creo que sólo puede tomar valores positivos). El parámetro **Eje**, indica la dirección de la base perpendicular al plano en el se dibuja el texto, además de la dirección dentro de dicho plano en la que queda alineado el texto.

Cuando el parámetro **Eje** vale:

- 1 el texto se alinea en la dirección del eje 2
- 2 el texto se alinea en la dirección del eje 3
- 3 el texto se alinea en la dirección del eje 1

Especificación del color de dibujo por defecto:

```
RGB(numero_Red, numero_Green, numero_Blue [, numero_Alpha])
```

Esta instrucción cambia el color de dibujo por defecto, de forma que después de introducir esta instrucción los elementos que se dibujen tomarán el citado color a no ser que se especifique el parámetro **RGB(numero_Red, numero_Green, numero_Blue [, numero_Alpha])** en la propia instrucción de dibujo del elemento.

El parámetro especial, `RGB(numero_Red, numero_Green, numero_Blue [, numero_Alpha])`, se utiliza para especificar el color del objeto utilizando el standard RGB. El único parámetro que necesita explicación es `numero_Alpha`, que indica la opacidad del elemento. Un valor de 1 indica que el objeto es completamente opaco: valores inferiores a 1 representan objetos con diferente grado de transparencia.

Para borrar el último objeto dibujado:

```
UNDRAW_LAST_OBJECT
```

Esta instrucción puede utilizarse varias veces seguidas.

Para guardar una imagen de la ventana gráfica:

```
SAVE_GRAPHIC_OUTPUT("nombre_fich.ext")
```

Produce un fichero con una imagen del contenido actual de la Salida Gráfica. El tipo de fichero gráfico depende de la extensión `ext` utilizada (`ext` = jpg, jpeg, png, bmp, gif (no LZW), tiff (no LZW), etc). Para que esta funcionalidad esté disponible la librería gráfica *ImageMagick* debe estar instalada, si no puede haber problemas.

Para guardar una animación o secuencia de imágenes de la ventana gráfica:

```
SAVE_GRAPHIC_OUTPUT_ANIMATION(Activada={0,1,2} [,"nombre_fich_base$t$.ext"])
```

Hace que durante la ejecución de las instrucciones de integración se genere un fichero gráfico con por cada instante en que se dibuja la salida gráfica. La denominación del fichero es análoga a la de la instrucción anterior, es decir el tipo de fichero gráfico depende de la extensión `ext` utilizada (`ext` = jpg, jpeg, png, bmp, gif (no LZW), tiff (no LZW), etc). El nombre del fichero que se escribe queda determinado por la sustitución de la cadena `$t$` por el instante de tiempo.

Si la opción `Activada` es 0 se desactiva la escritura de animaciones, si es uno `t` se escribe en formato 'real' (p.e.: si `t=10.1` entonces `$t$=0010.10`) y si es 2 en vez de el tiempo se sustituye el número de imagen, teniendo en cuenta posibles animaciones anteriores.

La cadena `$t$` puede ir al principio (`$t$nombre_fich_base.ext`) en el medio (`nombre_fich_base$t$.ext`) o al final (`nombre_fich_base.ext$t$`).

Ejemplos:

```
SAVE_GRAPHIC_OUTPUT_ANIMATION(2,"$t$elipsoide.png")
SAVE_GRAPHIC_OUTPUT_ANIMATION(2,"elipsoide$t$.bmp")
```

Para que esta funcionalidad esté disponible la librería gráfica **ImageMagick** debe estar instalada, si no puede haber problemas.

En el caso de que la librerías anteriores esten instaladas, junto con los 'delegates' mepeg2encode y mpeg2decode esten instaladas será posible generar una animación 'mpg' introduciendo la siguiente instrucción en la línea de comandos:

```
convert *nombre_fich_base.ext  nombre_fich_base.mpg
```

Análogamente pueden realizarse animaciones 'mng'y 'gif'con:

```
convert *nombre_fich_base.ext  nombre_fich_base.gif
```

```
convert *nombre_fich_base.ext  nombre_fich_base.mng
```

Lamentablemente la compresión 'LZW' en 'gif' está patentada, lo que sin la licencia apropiada implica que las animaciones ocupan un gran tamaño. Respecto al formato 'mpg' aunque sus características son buenas está poco soportado por las aplicaciones más comunes del mercado.

3.3.8 Cambio del punto de vista y del tipo de proyección.

Cambio del Punto de Vista (Visual).

La instrucción **VISUAL** permite cambiar el punto de vista, mejor dicho el punto del espacio desde el que se observa la escena. Además permite controlar el punto de la escena hacia el que se se mira, y la distancia desde la que se mira¹.

Sintaxis:

```
VISUAL(psi, theta, [ distancia, [ Delta_x, Delta_y, Delta_z ] ] )
```

Donde:

- **psi, theta, distancia**: Números reales (los primeros en grados sexagesimales) que representan en coordenadas esféricas respecto de la **Referencia Absoluta** (sección 3.3.6) **abs**, la posición del punto del espacio desde el que se observa.

¹Al menos si la proyección es cónica, pues si es ortográfica la variable distancia se interpretará como un factor que permite controlar el tamaño y que es inversamente proporcional a este.

- **Delta_x, Delta_y, Delta_z:** Cuando dichos parámetros son nulos el punto hacia el que se mira se sitúa sobre el origen de la referencia **abs**, punto 0, si no indican la variación de la posición del punto desde el que se observa respecto del punto 0 y en coordenadas cartesianas en la base **xyz**.

Por defecto el programa utiliza la dirección de observación que se obtendría al ejecutar esto es, se observa el mecanismo desde la dirección 1 de la Base **xyz**. Esto hace que la dirección 1 (**x**) apunte hacia el usuario, la dirección 2 (**y**) sea horizontal y apunte hacia la derecha del usuario, y la dirección 3 (**z**) apunte hacia arriba.

EJEMPLO:

```
VISUAL( 25, 15)
VISUAL( 25, 15, 30)
VISUAL( 25, 15, 10, 0, 10)
```

Tipos de perspectiva o proyección gráfica.

También existe la posibilidad de utilizar una perspectiva de tipo cónica, según la cual los objetos son más pequeños cuanto más alejados están. Alternativamente puede utilizarse una perspectiva ortográfica de forma que el cambio del tamaño de los objetos con la distancia desaparece. Las instrucciones son:

Sintaxis:

CONIC_PERSPECTIVE

Sintaxis:

ORTHOGRAPHIC_PERSPECTIVE

3.3.9 Integración del Sistema de D.A.E.s

El programa detecta automáticamente los Sistemas de Ecuaciones Diferenciales que definen la evolución temporal de los Sistemas o problemas planteado. Por tanto no es necesario especificarlas explícitamente.

En realidad lo que hace el *núcleo* es Integrar con respecto al tiempo todas aquellas variables para las cuales haya sido definida explícitamente su **derivada con respecto al tiempo** (sección 3.3.1) , a no ser de que la variable en cuestión quede definida a través de un/a:

- **Instrucción de asignación** (sección 3.3.1) que le asigne una determinada Expresión a través de la cual calculará su valor.
- **Sistema de ecuaciones Lineales** (sección 3.3.4) en cuyo caso tomará su valor a través de la resolución o aplicación de éste.
- **Sistema de ecuaciones No Lineales** (sección 3.3.3) en cuyo caso tomará su valor a través de la resolución o aplicación de éste.

NOTA: El *núcleo* evita que el usuario haga que una variable tenga diferentes definiciones; por ejemplo: a través de una Instrucción de asignación y a través de un Sistema de Ecuaciones Lineales. De esta forma es posible mantener **coherentes** (sección 3.2) , en todo instante de tiempo, los valores de las variables (numéricos y simbólicos) con las expresiones o sistemas de ecuaciones que las determinan.

Están implementados diferentes algoritmos para la Integración Numérica del Sistema de Ecuaciones Diferenciales. Todos ellos contemplan la posibilidad de escribir los diferentes Estados del Sistema, calculados a lo largo de la Integración, en el **Fichero de Estado** (sección 3.3.10) . Complementariamente existe un conjunto de instrucciones que permiten controlar determinados aspectos del Fichero de Estado, así como utilizarlo como entrada para la **visualización de una simulación previamente calculada** (sección 3.3.10) .

Método de *Euler*.

La instrucción EULER realiza la integración del Sistema de Ecuaciones Diferenciales mediante el método de Euler.

Sintaxis:

```
EULER(delta_t_integracion, duracion_integracion,
      delta_t_visualizacion, cada_cuanto_escribir_estado)
```

Donde:

- **delta_t_integracion**: Número Real que indica el incremento de tiempo con el que se realiza la integración. Se recomienda que para velocidades generalizadas de orden unidad se tomen incrementos de tiempo del orden de una centésima como máximo.
- **duracion_integracion**: Número Real que indica la duración intervalo de tiempo a lo largo del cual se desea realizar la integración.

- **delta_t_visualizacion**: Número Real que indica cada cuánto tiempo se va a visualizar la escena (esto permite observar la evolución en el tiempo). Un valor de 0 hace que no se visualice la evolución de la representación gráfica del sistema a lo largo de la simulación. Lo más normal es tomar `delta_t_dibujo >= delta_t_grafico`.
- **cada_cuanto_escribir_estado**: Número Entero que indica cada cuántas iteraciones de integración se va a escribir en el **Fichero de Estado** (sección 3.3.10) el *estado* de la simulación. Si el modo de sobreescritura del Fichero de Estado está activo, el Fichero de Estado activo se sobrescribe perdiéndose su contenido (si este existía). Si el modo de sobreescritura está desactivado, el conjunto de estados escrito se añade al final del Fichero de Estado actual. Si `cada_cuanto_escribir_estado = 0`, no se escriben los estados en el Fichero de Estado.

Método de *Runge Kutta* de Orden 4.

La instrucción RK4 realiza la integración del Sistema de Ecuaciones Diferenciales mediante el popular método *Runge Kutta de Orden 4*. Computacionalmente hablando, este método es aproximadamente 4 veces más costoso que el método de Euler, siempre y cuando el incremento temporal que se utilice para la Integración sea el mismo. La ventaja es que el error cometido con este método en cada paso de Integración es del orden del Incremento Temporal de Integración elevado a la quinta, frente al de Euler que comete un error del orden del Incremento Temporal de Integración elevado al cuadrado. Por tanto el éste método es mucho más preciso.

Sintaxis:

```
RK4(delta_t_integracion, duracion_integracion,
    delta_t_visualizacion, cada_cuanto_escribir_estado)
```

Donde el significado de los parámetros es el mismo que para la **Instrucción** (sección 3.3.9) **EULER**.

Método de *Runge Kutta* de Orden 5 Adaptativo.

La instrucción RK5_ADAPTATIVO realiza la integración del Sistema de Ecuaciones Diferenciales, mediante un método de Runge Kutta de Orden 5 Adaptativo. Tiene la ventaja, frente a los anteriores, de que este método ajusta automáticamente el incremento temporal con que se realiza la integración de forma que se garantiza

que el error cometido en cada paso de integración es suficientemente pequeño, con lo cual la calidad de la Integración del Sistema de Ecuaciones Diferenciales queda garantizada. Computacionalmente hablando tiene la desventaja de que el método es aproximadamente 6 veces más lento que el método de Euler, esta desventaja puede verse compensada en parte por el hecho de que el incremento temporal es siempre lo más grande posible para el nivel de precisión requerido.

NOTA: Actualmente el nivel de precisión no es un parámetro accesible desde la instrucción, este queda fijado en el fichero de inicialización **3D_MEC.INI** (sección 4) .

Sintaxis:

`RK5_ADAPTATIVO(duración, delta_t_gráfico, cada_cuanto_escribir_estado)`

Donde el significado de los parámetros es el mismo que para las **Instrucciones** (sección 3.3.9) **EULER** y **RK4** (nótese la ausencia en este método del parámetro `delta_t_integracion`).

3.3.10 El Fichero de Estado.

Los Ficheros de Estado son empleados por el *núcleo* para guardar un registro de los diferentes *Estados* en los que se encuentra el *Sistema* a lo largo de la *Simulación*.

3D_Mec puede importar o leer el *estado* de uno de estos ficheros, lo cual permite reanudar una simulación con diferentes parámetros a partir de alguno de los estados guardados. También es posible visualizar y analizar simulaciones previamente calculadas. Para sistemas medianamente complejos la velocidad de lectura-visualización es generalmente superior a la que se obtiene mediante el cálculo-visualización.

El formato del Fichero de Estado es el siguiente:

- La primera línea contiene, en formato ASCII, el nombre de todas las *Variables de Estado* definidas en el punto del *Fichero de Entrada de Datos* en el que se introduce la primera instrucción que escriba en el *Fichero de Estado*. Los nombres de estas variables están separados por tabuladores.
- Cada una de las siguientes líneas contiene, en formato ASCII, el valor Numérico de las *Variables de Estado* en cada uno de los instantes en que se escribe el *estado*. Estos valores están ordenados como en la primera línea y están separados por tabuladores.

De esta forma *Fichero de Estado*, puede ser importado por la mayoría de procesadores de texto, hojas de cálculo, y paquetes típicos de representación gráfica de funciones y/o proceso de datos, permitiendo el postproceso. De esta forma 3D_Mec se convierte en una aplicación abierta.

Nombre del Fichero de Estado.

La orden `SET_STATE_FILE` selecciona el *Fichero de Estado* a utilizar. A partir del momento en que se introduce todas las operaciones de lectura o escritura de *Estados* tienen lugar sobre el fichero especificado.

Sintaxis:

```
SET_STATE_FILE(nombre_fichero_estado)
```

Donde:

- `nombre_fichero_estado`: Es el nombre del *fichero de estado*.

Si el fichero existe se puede leer los diferentes *Estados* que contiene, escribir nuevos estados o sobrescribirlo con nuevos estados. Si el fichero de estado especificado no existe solo es posible la escritura de nuevos estados.

La orden en sí no crea (en el caso de que no exista), ni modifica el fichero de estado seleccionado. Solo las **instrucciones que permiten escribir en este fichero** (sección 3.3.10) serán responsables en alguna manera de su creación o modificación. Es decir, es imposible modificar, sobrescribir o borrar este fichero por el solo hecho de usar ésta instrucción.

Utilizando esta instrucción varias veces es posible utilizar distintos ficheros de estado para escribir o leer diferentes etapas de la *Simulación*.

Si no se utiliza esta orden se utiliza por el *fichero de estado* por defecto (normalmente “ESTADO.MEC”). El nombre por defecto se especifica en el **fichero de inicialización** (sección 4) “3D_MEC.INI”.

Escritura de “Un”Estado en el Fichero de Estado.

La instrucción `WRITE_STATE` escribe el *Estado Actual* en el *Fichero de Estado Actual*.

Sintaxis:

WRITE_STATE

Si el *fichero de estado* actual está vacío o se acaba de activar el **modo de sobrescritura** (sección 3.3.10) del *fichero de estado* entonces, después de ejecutar la instrucción el *fichero de estado* tendrá dos únicas líneas:

- la primera línea del *fichero de estado* contiene los nombres de las *Variables de Estado* separados por tabuladores.
- la segunda línea del *fichero de estado* contiene, en el mismo orden, los valores numéricos de las *variables de estado* separados por tabuladores.

Si no, la orden hace que:

- se escriba una nueva línea , al final del fichero, con los valores numéricos de las *variables de estado* separadas por tabuladores. El orden en que se escribe el valor de las variables es correspondiente al de la primera línea del *fichero de estado* actual.

NOTA (¡PELIGRO!): Debe tenerse en cuenta que si se acaba de activar el modo de sobrescritura del *fichero de estado* ésta orden hará que se pierda cualquier información que contenga el *fichero de estado* actual.

NOTA: Existen otras **Instrucciones que escriben en el Fichero de Estado** (sección 3.3.10) .

Lectura del Fichero de Estado. Animación del Sistema.

La instrucción READ_STATE lee uno o varios Estados del Fichero de Estado, y actualiza completamente el *núcleo* para mantener la **coherencia** (sección 3.2) con el/los Estado/s leído/s.

La instrucción puede utilizarse para realizar una animación del movimiento del *Sistema*.

Sintaxis:

```
READ_STATE(numero_estado_inicial, [numero_estado_final, visualizar_cada])
```

Donde:

- **numero_estado_inicial**: Número entero que indica el número de orden, dentro del Fichero de Estado Actual, que ocupa el Estado con que se desea actualizar el *sistema*. Si solo se especifica éste parámetro (el resto son opcionales) el estado se actualiza con este Estado.
- **numero_estado_final**: Número Entero que indica el numero de orden, dentro del Fichero de Estado Actual, que ocupa el último Estado con el que se actualiza el Sistema. Este parámetro es opcional y debe utilizarse conjuntamente con el siguiente para animar gráficamente el esquema del sistema.
- **visualizar_cada**: Número Entero que indica cada cuantos Estados, de los comprendidos entre **numero_estado_inicial** y **numero_estado_final** se actualiza la *Salida Gráfica*.

Activar y Desactivar el modo de sobreescritura del Fichero de Estado.

La instrucción **STATE_OVERWRITE_ON**, activa el modo de sobreescritura del fichero de estado. Con este modo activo el *Fichero de Estado Actual* se sobreescribe en el momento en que se utilice una orden que tenga como efecto **escribir algo** (sección 3.3.10) en el *fichero de estado*. Por tanto si el *fichero de estado* actual no está vacío, se perderá su contenido cuando se realice sobre este la primera operación de escritura. Por razones de seguridad en la manipulación de Ficheros de Estado después de toda operación de escritura sobre el *fichero de estado*, se desactiva el modo de sobreescritura del *fichero de estado*. Además, y por defecto, el programa comienza siempre con el modo de sobreescritura desactivado.

Sintaxis:

STATE_OVERWRITE_ON

La instrucción **STATE_OVERWRITE_OFF** desactiva el modo de sobreescritura del *fichero de estado*. Con ello se consigue que todas las instrucciones que escriben en el *fichero de estado* lo hagan a continuación del último *estado* que haya sido escrito.

Sintaxis:

STATE_OVERWRITE_OFF

NOTA: Teóricamente no es necesario utilizar esta instrucción.

Otras Instrucciones que escriben en el Fichero de Estado.

A parte de la instrucción `WRITE_STATE` hay otras instrucciones que no han sido comentadas en este apartado y que permiten escribir en el *Fichero de Estado Actual*. Las instrucciones encargadas de **Integrar el Sistema de Ecuaciones Diferenciales** (sección 3.3.9) permiten, entre otras cosas, escribir sucesivamente diferentes *estados* de la *Simulación* en el *fichero de estado*.

3.3.11 Las Instrucciones STOP y PAUSE

La instrucción `STOP` detiene la ejecución de instrucciones y reinicializa el *núcleo* situando a este en el **estado STOP** (sección 3.1) .

Sintaxis:

`STOP`

La instrucción `PAUSE` detiene la ejecución de instrucciones, y sitúa al *núcleo* en el **estado PAUSE** (sección 3.1) .

Sintaxis:

`PAUSE`

Una vez detenida la ejecución de instrucciones, es posible continuar la ejecución de instrucciones a partir del punto donde se detuvo. Para ello basta con seleccionar la entrada de menú adecuada dentro del *interface* de 3D_Mec.

Además el *interface* de 3D_Mec permite la detención y continuación de la ejecución seleccionando la entrada adecuada en el menú. Antes de la detención se termina de ejecutar la instrucción en curso, la única excepción a esta regla son las instrucciones de **Integración** (sección 3.3.9) , que pueden ser interrumpidas y reanudadas justo después de que procese el paso elemental de integración correspondiente al momento de la interrupción.

3.4 Exportar

3.4.1 Exportar a MatLab

Es posible exportar a Matlab²MatLab es una software registrado por sus propietarios. del problema planteado.

Para ello puede utilizarse la instrucción:

```
EXPORT_TO_MATLAB
```

Básicamente se exporta el sistema de variables (las escalares) junto con las ecuaciones que las definen y sistemas de ecuaciones que las relacionan, que están definidos en el núcleo de 3D_Mec en el momento de ejecutar la orden, a un conjunto de ficheros `*.m` con la sintaxis de matlab, que permiten definir la mismas variables y resolver los mismos sistemas de ecuaciones.

Normalmente esta instrucción se ejecutará hacia el final del listado de instrucciones, antes de la integración, si es que se realiza.

Los ficheros exportados pueden apoyarse en algunas funciones generales Matlab que se encuentran en el directorio de configuración `3d_mec\matlab_macros`. La información más actualizada sobre la exportación a matlab debe encontrarse en forma de fichero `readme.txt` dentro del mismo directorio.

NOTA: 3D_Mec no modifica el nombre de las variables, con lo cual habrá de tenerse en cuenta que alguno de ellos puede ser incompatible con matlab. Por ejemplo, en los ejemplos se ha utilizado el caracter prima (') detrás del nombre de una variable para representar su derivada, este símbolo es un operador en Matlab lo cual puede dar, en general, problemas.

De momento no se introduce ninguna instrucción que permita simular el sistema, ni resolver los sistemas de ecuaciones, ni representar los diferentes elementos gráficos. Estas cuestiones que deben ser tratadas por el usuario.

²Matlab

Capítulo 4

Configuración. Internacionalización.

4.1 Configuración

Existen ciertas posibilidades de configuración de 3D_Mec. Actualmente estas posibilidades se controlan a través del fichero “**3d_mec.conf**”. El fichero contiene comentarios que indican como modificar las diferentes opciones. La modificación de este fichero es solo apropiada para usuarios expertos.

4.2 Internacionalización

A partir de la versión 1.99beta19 puede elegirse el inglés como sintáxis de las instrucciones de 3d_mec. En un futuro próximo la sintáxis en inglés será la sintáxis por defecto. En este sentido se distribuye el manual de usuario en dos versiones: con la sintaxis de las instrucciones en inglés y con esta en castellano (**man_usr-es.pdf** y **man_usr-en.pdf**). Lo mismo se ha hecho con el directorio de ejemplos, ahora se distribuyen dos directorios, **examples-en** y **examples-es**, cada uno con la sintaxis respectiva.

Los usuarios que deseen utilizar la sintáxis en inglés deberán borrar el fichero **instruc.conf** y copiar en su lugar el fichero **instruc-en.conf** o bien todos ellos bajo el directorio **3d_mec/conf**. Es posible que en el futuro aparezcan más traducciones de las instrucciones que recibirán nombres **instruc-ver-idioma.conf** o simplemente **instruc-idioma.conf**.

La correspondencia entre nombres de instrucciones para diferentes idiomas, aunque es bastante trivial, puede determinarse observando el contenido de los ficheros `instruc-*.conf` para los distintos idiomas.

A continuación se muestra la tabla de traducción que permite pasar de la antigua versión a la nueva versión.

REM	REM
PAUSA	PAUSE
PUNTO	POINT
REF	REF
BASE	BASE
ECUAC_LIN	LINEAR_EQUATION_SYSTEM
ECUAC_NO_LIN	NON_LINEAR_EQUATION_SYSTEM
VISUAL	VISUAL
ESCALA	SCALE
MUESTRA	PRINT
EULER	EULER
RK4	RK4
RK5_APTATIVO	RK5_ADAPT
LEE_ESTADO	READ_STATE
ESCRIBE_ESTADO	WRITE_STATE
FICHERO_ESTADO	SET_STATE_FILE
ACTIVA_SOBREESCRITURA_ESTADO	STATE_OVERWRITE_ON
DESACTIVA_SOBREESCRITURA_ESTADO	STATE_OVERWRITE_OFF
STOP	STOP
DIBUJA_ESFERA_OpenGL	DRAW_SPHERE
DIBUJA_ELIPSOIDE_OpenGL	DRAW_ELLIPSOID
DIBUJA_CILINDRO_OpenGL	DRAW_CYLINDER_P
DIBUJA_CONOP_OpenGL	DRAW_CONE_P
DIBUJA_VECTOR_OpenGL	DRAW_VECTOR
DIBUJA_REFERENCIA_OpenGL	DRAW_REFERENCE
DIBUJA_LINEA_OpenGL	DRAW_LINE
DIBUJA_PARALELEPIPEDO_OpenGL	DRAW_PARALLEPIPED
DIBUJA_CILINDRO_OpenGL	DRAW_CYLINDER
DIBUJA_ARCO_CILINDRO_OpenGL	DRAW_CYLINDER_ARC
DIBUJA_RECTANGULO_OpenGL	DRAW_RECTANGLE
DIBUJA_TEXTO_OpenGL	DRAW_TEXT
DESTRUYE_ULTIMO_OBJETO_OpenGL	UNDRAW_LAST_OBJECT
DIBUJA_LINEA_OpenGL	DRAW_LINE
DIBUJA_HIPERBOLOIDE_CILINDRICO_OpenGL	DRAW_CYLINDRIC_HYPERBOLOID
EXPORTAR_A_MATLAB	EXPORT_TO_MATLAB
GUARDA_ANIMACION_SALIDA_GRAFICA_OpenGL	SAVE_GRAPHIC_OUTPUT_ANIMATION
GUARDA_SALIDA_GRAFICA_OpenGL	SAVE_GRAPHIC_OUTPUT
PERSPECTIVA_CONICA	CONIC_PERSPECTIVE
PERSPECTIVA_ORTOGRAFICA	ORTHOGRAPHIC_PERSPECTIVE
DIBUJA_CONO_OpenGL	DRAW_CONE
DIBUJA_ARCO_CILINDRO_VACIO_OpenGL	DRAW_EMPTY_CYLINDER_ARC
DIBUJA_ARCO_ESFERA_OpenGL	DRAW_SPHERE_ARC
DIBUJA_ARCO_ESFERA_VACIO_OpenGL	DRAW_EMPTY_SPHERE_ARC
FIN_ECUAC_LIN	END
FIN_ECUAC_NO_LIN	END

Sólo usuarios expertos:

Las traducciones son fácilmente automatizables en entornos LINUX con la utilidad `replace` (proveniente **MySQL**) instalada, para ello se proporciona el script `spanish_to_english.bash`, dentro del directorio de configuración `'conf'`. En los sistemas linux redhat se encuentra frecuentemente instalada o queda instalada al instalar las RPM correspondientes de MySQL. La utilidad `replace.exe` se proporciona con la distribución en entornos windows, aunque para ejecutar el script anterior es necesario un shell `bash` (lo que de hecho restringe su utilización a usuarios en linux o bien usuarios expertos en windows, [puede utilizarse el entorno Cygwin para ello <http://www.cygwin.com>]).

Capítulo 5

Instalación

La instalación no debe presentar ningún problema.

En Windows basta con ejecutar el archivo de la instalación y seguir las instrucciones. En linux basta instalar el fichero de extensión `rpm`, para ello puede ejecutarse:

```
rpm -i *3d\_mec*.rpm
```

como root, aunque esto normalmente implicará tener un sistema específico linux, con lo cual es conveniente leer las instrucciones correspondientes en el lugar de descarga.

Puedes descargar las últimas versiones de la aplicación desde la página web de [3D_Mec](#).

Capítulo 6

Disponibilidad de 3D_MEC en Internet

El lugar base en Internet en el que se puede encontrar información sobre el programa: como conseguirlo (ftp), última versión disponible etc., es fácilmente accesible en

http://www.imem.unavarra.es/3d_mec

Problemas relacionados con el acceso a dicha página deben ser comunicados a la siguiente dirección e-mail: jros@unavarra.es

Capítulo 7

Limitaciones y evolución futura de 3D_MEC

Las únicas funciones a parte de la exponenciación que actualmente se incluyen en el programa son el seno y el coseno (suficiente para plantear cualquier problema de Mecánica) no obstante se tiene previsto ampliar el conjunto de funciones soportado por el programa.

La aplicación se ejecuta correctamente en Windows NT 4.0 y Windows 98 (esta ultima plataforma no se ha probado mucho). Aunque en el caso de NT4.0 si la tarjeta no tiene Drivers OpenGL propios es necesario instalar el service Pack 3 (..hmmm.. Patch 3) o superior.

En windows 95 hay problemas graves, debido a la mala implementación del standard OpenGL en este sistema operativo. Es posible que algunas plataformas Windows 95 con tarjetas 3D OpenGL con drivers OpenGL propios, no presenten esta problemática, creo que microsoft tiene publicado algún parche pero, ahora es difícil encontrarlo. Microsoft distribuía un fichero llamado [opengl95.exe](#) que contenía las librerías opengl para los sistemas Windows 95 que no lo contenían (anteriores a OSR2) pero creo que aun así presentaba algún bug.

Respecto a Windows NT3.51, no se ha probado pero en principio si dispone de drivers OpenGL apropiados, no debe haber problemas.

Capítulo 8

Comunicación de errores

Los errores que se encuentren en el funcionamiento de la aplicación pueden comunicarse vía correo electrónico a la siguiente dirección: jros@unavarra.es .

Se ruega enviar solamente errores reproducibles, es decir: determinado fichero de instrucciones 3D_MEC cuelga la aplicación al ser ejecutado. También se ruega que la descripción del error ocurrido sea clara, y que se intente depurar el fichero que produce el error , es decir: hacer el fichero lo más pequeño y claro posible, siempre y cuando reproduzca el error. También se sugiere que se compruebe previamente si la versión que produce el error es la última disponible.

Glosario

Sistema

Para el *núcleo* el *Sistema* está formado por el conjunto de variables y las relaciones impuestas sobre estas (expresiones, sistemas de ecuaciones lineales o no lineales y leyes de evolución temporal que están obligadas a satisfacer), además de los puntos, bases y referencias definidos. Los diferentes elementos gráficos también forman parte del sistema.

Simulación

Por lo que respecta al *núcleo* la *Simulación* consiste en el cálculo a lo largo del tiempo del *Estado* del *Sistema* y de otras variables definidas. Para ello se calcula la evolución de todas las variables definidas obligando a que se satisfagan todas las relaciones impuestas sobre las variables.

Estado del sistema

Por lo que respecta al *núcleo*, el *estado* del *sistema* en un punto determinado de la ejecución del flujo de instrucciones, está caracterizado por el valor de todas las variables elementales utilizadas para definir el sistema.

Variables de Estado

Variables que determinan el estado del sistema.

Fichero de Estado

Fichero utilizado para guardar un registro de *Estados* del Sistema. Este fichero almacena el nombre y el valor numérico de las *Variables de Estado*.

Estado Actual

En un momento concreto de la *Simulación* se denomina *Estado Actual* al *Estado* del *Sistema* en ese momento.

Fichero de Estado Actual

En un momento concreto de la *Simulación* se denomina *Fichero de Estado Actual* al *Fichero de Estado* activo en ese momento. Es decir es el *fichero de estado* sobre el que se realizan las operaciones de lectura-escritura en dicho momento.

Punto

En 3D_Mec se maneja un espacio euclideo de dimensión 3. Un punto representa de forma abstracta una posición determinada del espacio. 3D_Mec permite la definición de puntos. Es posible hacer referencia a los diferentes puntos tanto en instrucciones de dibujo como en expresiones. Esto facilita enormemente el manejo de expresiones y ecuaciones vectoriales, y el posicionamiento de los diferentes elementos gráficos en el espacio.

Base

En 3D_Mec se maneja un espacio euclideo de dimensión 3. Las bases representan un triedro de *vectores* ortonormales ordenados dextrógiramente, también puede decirse que una base representa una *orientación* en el espacio. 3D_Mec permite definir bases. De esta forma los cambios de bases quedan automatizados utilizando el operador “Llaves”. Esto permite que 3D_Mec maneje vectores y tensores, y que permita la asociación de bases con 3-tuplas y matrices 3x3. Existen diferentes instrucciones y funciones que pueden hacer referencia a las bases, facilitando enormemente el manejo de expresiones y ecuaciones vectoriales, y el posicionamiento de los diferentes elementos gráficos en el espacio.

Referencia

En 3D_Mec se maneja un espacio euclideo de dimensión 3. De un modo abstracto una referencia representa una posición y una *orientación* en el espacio respecto de la que se observa. En concreto 3D_Mec maneja dicho concepto permitiendo la definición de un referencial. Un referencial está formada por una *base* situada en un *punto* concreto del espacio. Debido a la similitud de conceptos, no se hace distinción entre ellos y se utilizará el nombre referencia para ambos. Existen diferentes instrucciones y funciones que pueden hacer referencia estás, facilitando el manejo de expresiones y ecuaciones vectoriales, y el posicionamiento de los diferentes elementos gráficos en el espacio.

Orientación

Para centrar ideas puede decirse que dos objetos indeformables tienen la misma orientación si su velocidad angular relativa es nula. Las *bases* de proyección son objetos matemáticos orientados y las referencias tienen la misma orientación que las bases utilizadas en su definición. 3D_Mec permite hacer referencia a una determinada orientación a través del nombre de una base o bien por el nombre de una referencia.

3-Tupla

3D_Mec utiliza un espacio vectorial euclideo de dimensión 3. En 3D_Mec una 3-tupla representa las componentes de un *vector* en una determinada

base de dicho espacio. 3D_Mec permite la definición de 3-tuplas, opcionalmente es posible asociarles una base de proyección por medio del operador “Llaves”.

En el álgebra matricial es común definir n-tuplas en espacios de dimensión n. Usualmente suele diferenciarse entre n-tuplas fila y n-tuplas columna. 3D_Mec no hace esta distinción, dada la definición de 3-tupla que utiliza. No obstante, debido al peso del álgebra matricial y para centrar ideas, puede pensarse que en 3D_Mec las 3-tuplas son 3-tuplas columna.

Vector

3D_Mec utiliza un espacio vectorial euclideo de dimensión 3. Un vector es un elemento de dicho espacio vectorial. 3D_Mec representa gráficamente los vectores de la forma convencional utilizada en física: mediante una flecha. Para diferenciar entre vectores y 3-tuplas puede decirse, por ejemplo, que la 3 tupla es un concepto más matemático mientras el vector es más físico, y que mientras que el vector es *sólo uno*, puede tener diferentes representaciones como 3-tuplas, tantas como bases de proyección se definan.

Matriz 3x3

3D_Mec utiliza un espacio vectorial euclideo de dimensión 3. En 3D_Mec es posible definir matrices 3x3, opcionalmente es posible asociarles una [base](#) de proyección por medio del operador “Llaves”. Una matriz 3x3 está formada por 3 [3-tuplas](#) ordenadas, cada una de ellas puede considerarse que representa las componentes de un determinado *vector*. Estableciendo una analogía puede decirse que una matriz 3x3 es a un *tensor* como una 3-tupla a un vector. Esto es, una matriz 3x3 representa la proyección de un tensor en una determinada base. Es decir una matriz-3x3 representa las 3 3-tuplas ordenadas correspondientes a las proyecciones de los 3 vectores que definen un tensor. Es por ello que en 3D_Mec es posible asociar una base a una matriz 3x3.

3D_Mec no hace distinción entre matriz y matriz transpuesta, dada la definición de matriz 3x3 que utiliza. No obstante, debido al peso del álgebra matricial y para centrar ideas, puede pensarse que en 3D_Mec las matrices 3x3 estan formadas por 3 filas y que cada una de ellas es una de las 3 3-tuplas ordenadas.

Debido a las definiciones de 3-tupla y matriz 3x3 que maneja 3D_Mec una matriz 3x3 únicamente puede multiplicarse por su derecha por una 3-tupla (Notese la similitud con la operación del álgebra matricial: matriz 3x3 por 3-tupla columna).

Tensor

3D_Mec utiliza un espacio vectorial euclideo de dimensión 3. En 3D_Mec un tensor se define como un conjunto ordenado de 3 *vectores* de dicho espacio. En mecánica el tensor utilizado más habitualmente es el tensor de inercia, que representa de cierta forma una propiedad asociada a la cantidad y a la distribución de masa de un determinado sólido en una determinada orientación. Para entender que es un tensor en 3D_Mec, puede decirse que un vector es a una 3-tupla, como un tensor es a una matriz-3x3.

Bibliografía

- [1] Joaquim Agulló Batlle, et al.; **MECÁNICA, de la partícula y del sólido rígido**; Publicaciones OK Punt. (e.t.s.e.i.b.); ISBN 84-920850-2-9; 1996.
- [2] Press, William H., et al.; **Numerical Recipes (in FORTRAN) 2nd Ed.**; Press Syndicate of the University of Cambridge; ISBN 0-521-43064-X; 1992.
- [3] http://www.inem.unavarra.es/3d_mec .